Geometry and Contraction:
A Riemannian Framework for Safe Robot Motion Learning

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

M.Sc.

Hadi Beik-Mohammadi

Tag der mündlichen Prüfung:	28.07.2025
Erster Gutachter:	Prof. Dr. Gerhard Neumann
Zweiter Gutachter:	Prof. Dr. Jens Kober

Abstract

Robot motion generation remains a critical challenge, particularly in unstructured and dynamic environments where ensuring safety and adaptability is essential for robust performance. In this thesis, we introduce a geometric framework for motion generation that systematically addresses kinematic and dynamical aspects as distinct yet interrelated components. While the study of motion kinematics and dynamics has been extensively investigated, our approach is distinguished by formulating both within the geometry of Riemannian manifolds. This perspective enables a more intrinsic representation of motion, leveraging the geometric properties of the underlying data manifold to ensure consistency and adaptability in dynamic environments.

At the kinematic level, we introduce the concept of geodesic motion skills. Human demonstrations are modeled as smooth trajectories on a Riemannian manifold, where the intrinsic geometry defines optimal paths. By employing a variational autoencoder to learn a latent representation of these motion manifolds, our approach synthesizes full-pose end-effector trajectories through the computation of intrinsic shortest paths namely geodesics. Moreover, by dynamically modulating the underlying metric, the method inherently supports real-time obstacle avoidance. This framework is further extended to the robot's joint space, facilitating multiple-limb obstacle avoidance and accommodating multiple-solution tasks.

At the dynamical level, we propose neural contractive dynamical systems (NCDS), which integrate principles from contraction theory into neural network architectures. In this formulation, system dynamics are modeled as time-invariant vector fields whose stability is ensured through inherent contraction properties. An injective variant of the variational autoencoder is utilized to handle high-dimensional demonstration data while preserving these stability guarantees. By extending the approach to incorporate the mathematical structure of Lie groups, our method accurately captures complex orientation dynamics, further enhancing its applicability. Additionally, the integration of Riemannian implicit distance functions allows for adaptive modulation of the dynamics, effectively blending the learned Riemannian geometry with obstacle avoidance strategies to ensure safety around obstacles and regions of uncertainty.

In conclusion, this work demonstrates that geometric principles serve as a fundamental framework for advancing robot learning. By rigorously applying concepts from differential geometry, this research has revealed the inherent structure underlying human motion and has provided a systematic method for encoding complex movement patterns. This approach has enabled the unification of diverse learning methodologies into a coherent framework that more effectively captures the subtleties of motion. The insights gained highlight the critical role of geometry in deepening our understanding of robot behavior.

Kurzfassung

Die Erzeugung von Roboterbewegungen bleibt eine wesentliche Herausforderung, insbesondere in unstrukturierten und dynamischen Umgebungen, in denen die Gewährleistung von Sicherheit und Anpassungsfähigkeit für eine robuste Leistung unerlässlich ist. In dieser Dissertation führen wir ein geometrisches Framework für die Bewegungserzeugung ein, das kinematische und dynamische Aspekte systematisch als voneinander unterscheidbare, aber miteinander verknüpfte Komponenten betrachtet. Obwohl die Untersuchung der Bewegungskinematik und -dynamik bereits umfassend untersucht wurde, zeichnet sich unser Ansatz dadurch aus, dass beide basierend auf der Geometrie von Riemannschen Mannigfaltigkeiten formuliert werden. Diese Perspektive ermöglicht eine intrinsischere Darstellung von Bewegung, indem sie die geometrischen Eigenschaften der zugrunde liegenden Datenmannigfaltigkeit nutzt, um Konsistenz und Anpassungsfähigkeit in dynamischen Umgebungen sicherzustellen.

Auf kinematischer Ebene führen wir das Konzept der geodätischen Bewegungsfertigkeiten ein. Menschliche Demonstrationen werden als glatte Trajektorien auf einer Riemannschen Mannigfaltigkeit modelliert, wobei die intrinsische Geometrie optimale Pfade definiert. Durch den Einsatz eines variational Autoencoders zum Erlernen einer latenten Darstellung dieser Bewegungsmannigfaltigkeiten synthetisiert unser Ansatz vollständige Endeffektortrajektorien durch die Berechnung intrinsischer kürzester Wege den sogenannten Geodäten. Darüber hinaus unterstützt das Verfahren von Natur aus Echtzeithindernisvermeidung durch dynamische Modulation der zugrunde liegenden Metrik. Dieses Framework wird ferner auf den Gelenkraum des Roboters erweitert, wodurch Ganzkörperhindernisvermeidung ermöglicht wird und Aufgaben mit mehreren Lösungen berücksichtigt werden.

Auf dynamischer Ebene schlagen wir Neural Contractive Dynamical Systems (NCDS) vor, die Prinzipien der Kontraktionstheorie in neuronale Netzwerkarchitekturen integrieren. Dabei werden Systemdynamiken als zeitinvariante Vektorfelder modelliert, deren Stabilität durch inhärente kontraktive Eigenschaften sichergestellt wird. Eine injektive Variante des variational Autoencoders wird eingesetzt, um hochdimensionale Demonstrationsdaten zu verarbeiten, wobei die Stabilitätsgarantien erhalten bleiben.

Durch die Erweiterung des Ansatzes um die mathematische Struktur von Lie-Gruppen erfasst unsere Methode komplexe Orientierungsdynamiken präzise und vergrößert somit ihren Anwendungsbereich. Zusätzlich ermöglicht die Integration von Riemannschen impliziten Distanzfunktionen eine adaptive Modulation der Dynamik, wodurch die erlernte Riemannsche Geometrie effektiv mit Hindernisvermeidungsstrategien kombiniert wird, um Sicherheit in der Nähe von Hindernissen und in Unsicherheitsbereichen zu gewährleisten.

Zusammenfassend demonstriert diese Arbeit, dass geometrische Prinzipien ein grundlegendes Framework für die Weiterentwicklung des Roboterlernens darstellen. Durch die rigorose Anwendung von Konzepten der Differentialgeometrie hat diese Forschung die zugrunde liegende Struktur menschlicher Bewegung offengelegt und einen systematischen Ansatz zur Kodierung komplexer Bewegungsmuster präsentiert. Dieser Ansatz ermöglicht die Vereinheitlichung unterschiedlicher Lernmethoden in einem kohärenten Framework, das die Feinheiten der Bewegung effektiver erfasst. Die gewonnenen Erkenntnisse unterstreichen die entscheidende Rolle der Geometrie für ein besseres Verständnis robotischen Verhaltens.

Acknowledgements

In loving memory of my mother, who walked beside me at the beginning of this journey but was not here to see its end. Her presence echoes in every step I take. Her strength, grace, and love continue to guide me.

To my father, whose quiet resilience, boundless belief, and countless sacrifices paved the way for all that I have accomplished. To my sister, whose love has been a constant in my life, no matter the distance. You have always believed in me, even when I could not see the path forward. Your strength, compassion, and wisdom have anchored me during the hardest times and inspired me more than you know. Thank you for your unconditional support, and for always reminding me that I was never alone. And to my brother-in-law, Davood, your optimism, kindness, and uplifting spirit have been a quiet force of encouragement throughout the years. To my cousin Behnam, more like a brother than a cousin, thank you for always being by my side, for the laughter, the honesty, and the countless shared memories that kept me grounded and lighthearted, even during the darkest days. And to my friend Behnam E., whose friendship and wholehearted support never wavered despite the passing of time and the demands of life, thank you for your sincerity, your quiet strength, and your enduring belief in me. To Jasmin, who stood by me through some of the most difficult and defining moments of this journey – your empathy and strength meant more than words can convey.

To Leonel, my supervisor at Bosch, your encouragement, patience, and trust gave me the confidence to navigate every challenge. You always made time, always listened, and never stopped believing in what I was capable of, even when I was not sure myself. Working with you was a rare privilege, and I am grateful for your consistent support at every step.

To Søren, my second advisor, you are one of the most extraordinary people I have had the honor of working with. Your clarity of thought, generosity with time and ideas, and calm, grounded presence have left a lasting impression on me, both professionally and personally. I learned so much from the way you think, advise, and I will carry those lessons with me for years to come.

To Geri, my university supervisor, thank you for walking this path with me with such composure and thoughtful care. Your feedback, mentorship, and patience, especially during the more complex phases of the work, meant a great deal.

To Georgios, whose contribution to this thesis runs deep, your dedication, critical insight, and tireless involvement helped shape the core of this work. Your commitment have meant more than I can say.

To my dear friends and colleagues—Mathias, Navid, Jan, Aleks, Setareh, Onur, Michael, David, Farnoush, Mehrnoush, Ava, Pouria, Hari, and many more – thank you for the conversations, the laughs, the encouragement, and the beautiful memories that helped me keep going.

Contents

Notation

Throughout this thesis, the following notation and symbols will be used.

General notation

Scalars	italic Roman and Greek lowercase letters	x, α
Vectors	bold Roman lowercase letters	$\boldsymbol{t}$
Matrices	bold Roman uppercase letters	$\boldsymbol{R}$
State spaces	bold calligraphic Roman uppercase letters	$\boldsymbol{\mathcal{X}}$

In multidimensional sets of elements related to time series, the first index denotes time.

State Variables and Dynamics

$\boldsymbol{x} \in \mathbb{R}^D$	state variable in the data space
$\dot{\boldsymbol{x}} \in \mathbb{R}^D$	temporal derivative of the state
$\boldsymbol{z} \in \mathbb{R}^d$	latent variable in the reduced space
$\dot{\boldsymbol{z}} \in \mathbb{R}^d$	temporal derivative of the latent variable
$\boldsymbol{\varpi} \in \mathbb{R}^k$	conditioning variable (task-related context)
$\boldsymbol{R} \in \mathrm{SO}(3)$	rotation matrix (special orthogonal group)
$\boldsymbol{r} \in \mathfrak{so}(3)$	skew-symmetric vector in the Lie algebra
$\boldsymbol{q} \in \mathcal{S}^3$	quaternion on the 3-sphere

Mathematical Constructs

$\boldsymbol{J}(\boldsymbol{x})$	Jacobian matrix of the system
$\hat{\boldsymbol{J}}_f(\boldsymbol{x})$	regularized Jacobian ensuring contraction
$\boldsymbol{J}_\mu(\boldsymbol{x})$	Jacobian of the decoder
$\boldsymbol{\lambda}$	eigenvalues of the Jacobian
$\hat{\boldsymbol{\lambda}}$	regularized eigenvalues
$\delta\boldsymbol{x}$	virtual displacement vector
$\boldsymbol{G}(\boldsymbol{x})$	modulation matrix for obstacle avoidance
$\boldsymbol{E}(\boldsymbol{x})$	basis matrix for the local tangent space
$\boldsymbol{D}(\boldsymbol{x})$	diagonal scaling matrix

Riemannian Geometry and Manifolds

$\boldsymbol{M}(\boldsymbol{x})$	Riemannian metric / contraction metric
$\mathcal{M} \subset \mathbb{R}^D$	Riemannian manifold in the data space
$\Omega : \mathbb{R}^d \to \mathbb{R}^D$	mapping function from latent to ambient space
$S(x) : \mathbb{R}^d \to \mathbb{R}$	distance field encoding obstacle geometry
$\nabla S(x)$	gradient of the distance field, giving obstacle normals
$V(x)$	volume element induced by the metric

Neural Network and Loss Functions

$\mu(\boldsymbol{z}) : \mathbb{R}^d \to \mathbb{R}^D$	mean decoder function
$\sigma(\boldsymbol{z}) : \mathbb{R}^d \to \mathbb{R}^D_+$	standard deviation decoder function
$\mathcal{L}_{\text{vel}}$	velocity reconstruction loss
$\mathcal{L}_{\text{Jac}}$	Jacobian regularization loss
$\mathcal{L}_{\text{ELBO}}$	evidence lower bound loss for variational autoencoders
$\boldsymbol{J}_\theta(\boldsymbol{x})$	neural network representation of the Jacobian

Spaces and Lie Groups

$\mathcal{Z} \subset \mathbb{R}^d$	latent space of reduced representation
$\mathcal{X} \subset \mathbb{R}^D$	ambient data space
SO(3)	special orthogonal group of rotations
$\mathfrak{so}(3)$	Lie algebra of SO(3)
$\mathcal{S}^3$	3-sphere

Obstacle Avoidance and Safety

$n(x)$	normal vector of the obstacle surface
$\rho \in \mathbb{R}_+$	reactivity factor for obstacle modulation
$\beta(x)$	tangential modulation factor
$\alpha \in \mathbb{R}_+$	metric scaling factor for boundary alignment

Constants and Scalars

τ	contraction rate ensuring stability
ε	regularization term for negative-definiteness
I	identity matrix

1 Introduction

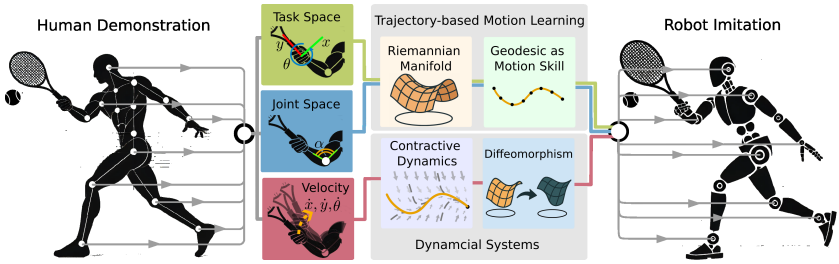


Figure 1.1: This thesis explores two distinct approaches to learning motion from human demonstrations: trajectory-based and dynamical system-based learning methods. These methods employ different data representations to accurately reconstruct motion.

Robot learning has gained interest recently because of its potential to endow robots with a repertoire of motion skills to operate autonomously or assist humans in repetitive, precise, and risk-averse tasks. In this thesis, we develop a geometric framework based on differential geometry, with a particular focus on Riemannian manifolds, to formulate models for kinematic and dynamic motion generation. The core contribution lies in the theoretical formulation, where the differential geometry provides a principled approach to modeling and analyzing motion. By leveraging the geometric properties of Riemannian manifolds, the framework enables a consistent treatment of both kinematic and dynamic aspects of motion, ensuring stability and adaptability in varying contexts. While the focus remains on the theoretical foundation, the proposed methods can be applied to various goal-directed robotic manipulation tasks.

1.1 Riemannian Geometry for Trajectory-Based Robot Skills

Focusing on the first category, trajectory-based motion is further divided into two subcategories: classic motion planners and movement primitives. Classic motion planners

generate obstacle-free continuous trajectories between start and goal configurations using search or sampling-based algorithms (Elbanhawi et al., 2014, Mohanan et al., 2018). Movement primitives, in contrast, are a modular abstraction of robot movements, where a primitive represents an "atomic action" or an "elementary movement", which are often designed using robot learning techniques (Schaal et al., 2003). A relevant distinction between these categories is that motion planners require a precise description of the robot workspace to generate a continuous path to the goal, while movement primitives rely on observed motion patterns to encapsulate the desired trajectory without an explicit model of the environment. If an unseen obstacle shows up in the robot workspace, motion planners often need to replan the entire trajectory from scratch, while most movement primitives require to be retrained, unless they are provided with via-point passing features.

In this thesis, we delve into the differential geometric underpinnings of motion generation by examining the trade-offs between traditional motion planning and movement primitives. Our approach is learning-based and reactive, integrating human demonstration data to model skills – akin to movement primitives – while simultaneously constructing a time-independent reference trajectory with built-in obstacle avoidance capabilities comparable to motion planners. Central to our methodology is the utilization of the rich geometric structure provided by Riemannian manifolds. By framing the motion generation problem within the context of differential geometry, we leverage the natural properties of smooth manifolds to capture the inherent variability and efficiency of human motion. In this framework, trajectories are not arbitrary curves but are defined as geodesics—paths that represent the most energy-efficient connection between points, dictated by the underlying geometry of the manifold. This formulation allows us to derive motion policies that naturally adapt to dynamic obstacles while preserving the essential characteristics of the demonstrated motion patterns.

Unlike previous works (Havoutis et al., 2013, Li et al., 2018), where skill manifolds are built from locally smooth manifold learning (Dollár et al., 2007), we leverage a Riemannian formulation. Specifically, we develop a variational autoencoder (VAE) that learns a Riemannian submanifold of $\mathbb{R}^3 \times \mathcal{S}^3$ from human demonstrations (Beik-Mohammadi et al., 2021). *We exploit geodesics (i.e. shortest paths) on this learned manifold as the robot motion generation mechanism, from which full-pose end-effector trajectories are generated.* These geodesics reproduce motions starting from arbitrary initial points on the manifold, and they can adapt to avoid unseen dynamic obstacles in the robot environment by redefining the ambient metric. Our approach can learn manifolds encoding multiple-solution tasks, from which novel (unobserved) robot motions may

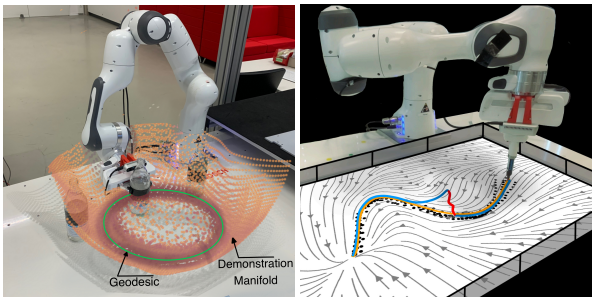


Figure 1.2: *Left*: From demonstrations we learn a variational autoencoder that spans a random Riemannian manifold. Geodesics on this manifold are viewed as motion skills. *Right*: Based on demonstrations, we learn a neural contractive dynamical system that guides the robot's movement.

naturally arise when computing geodesics. To systematically evaluate our approach, we propose the following hypothesis.

Hypothesis 1. *Let $\mathcal{M} \subset \mathbb{R}^d$ be a smooth manifold endowed with a Riemannian metric $\boldsymbol{M}$ that captures the intrinsic geometry of a set of human demonstrations, each well-approximated by a geodesic $\gamma : [0,1] \to \mathcal{M}$. We hypothesize that deriving* time-independent reference trajectories *from these geodesics replicates the essential kinematic features of the demonstrated motions. Furthermore,* reshaping *$\mathcal{M}$ using an ambient-space-induced metric $\boldsymbol{M}_{\mathcal{X}}$ naturally encodes obstacle avoidance, ensuring that while the reference trajectories preserve the demonstrated motion patterns, collision-free motion emerges as a* side product *of the manifold's geometry.*

Furthermore, inspired by the need of avoiding obstacles at any location of the robot body, we extended our geodesic motion skills concept to joint space skills (Beik-Mohammadi et al., 2023). To do so, we develop a new VAE architecture that integrates the robot's forward kinematics to access task space information at any point on the robot body. This new approach makes joint space skill adaptation possible, allowing the robot to simultaneously avoid unseen and dynamic obstacles and handle multiple-solution (a.k.a. multiple-mode) tasks in both task and joint spaces.

Hypothesis 2. *Let $\mathcal{M} \subset \mathbb{R}^n$ be a smooth Riemannian manifold representing skill demonstrations in the robot's joint space. Suppose the forward kinematics map $\kappa : \mathcal{M} \to \mathbb{R}^m$ provides task-space information for each joint configuration. We hypothesize that*

reshaping $\mathcal{M}$ *with an ambient-space-induced metric, in combination with geodesic-based trajectory generation, enables real-time collision-free motion at any point on the robot's body while accommodating multi-solution tasks in both joint and task spaces.*

We describe both methodologies with simple examples and extensively evaluate them on real tasks using a 7-DoF robot manipulator. The experiments showcase how our techniques allow a robot to learn and reconstruct motion skills of varying complexity. In particular, we show how our methods enable obstacle avoidance at task and joint space levels. Moreover, we provide experiments where the robot exploits multiple-solution tasks to effectively generate hybrid solutions without model retraining. Furthermore, we experimentally analyze how the latent space dimensionality impacts the quality of the geodesic motion generator, and how the robot behaves when controlled by geodesics crossing the manifold boundaries.

1.2 Contractive Dynamical Systems and Diffeomorphisms

Although geodesics can effectively reconstruct complex motion skills, they fail to consider the temporal dynamics and timing involved in the demonstrations. This limitation means that while the spatial aspects of the motion are captured, the crucial element of how the motion evolves over time is not captured. To address this challenge, this thesis then shifts focus to *stable dynamical systems* (shown as blocks with red connection path in Fig. 1.1) (Khansari-Zadeh et al., 2011a, Khansari-Zadeh et al., 2014, Neumann et al., 2015, Zhang et al., 2022b, Zhang et al., 2022a), introducing *neural contractive dynamical systems (NCDS)* that ensure stability through *contraction guarantees* (Lohmiller et al., 1998). Within the framework of differential geometry, we leverage the structure of *smooth manifolds* and coordinate transformations through *diffeomorphisms* to transport contraction properties from a latent space back to the original system, thereby enabling stability guarantees to hold across geometric representations.

Consider the robot in Fig. 1.2–*right*, which is trained from demonstrations (black dots) to execute a sinusoidal motion (orange). If the robot is pushed away (red perturbation) from its planned trajectory, then it should still be guaranteed to reproduce a motion similar to the demonstration pattern (blue). Manually designed robot movements could achieve such stability guarantees, but even skilled engineers struggle to hand-code highly dynamic motions (Billard et al., 2016). In contrast, learning robot dynamics from demonstrations has shown to be an efficient and intuitive approach for encoding highly dynamic motions into a robot's repertoire (Schaal et al., 2003). Unfortunately,

these learning-based approaches often struggle to ensure stability as they rely on the machine learning model to extrapolate in a controlled manner. In particular, as also shown in Fig. 1.3, it has proven surprisingly challenging to control the extrapolation behavior of neural network models especially when using Neural ODEs or multilayer perceptrons (MLPs) (Xu et al., 2021) which poses a significant obstacle to establishing reliable stability guarantees.

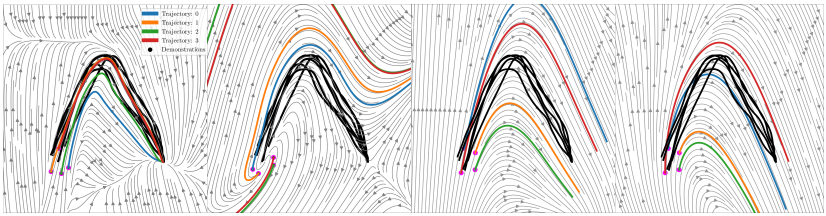


Figure 1.3: Path integrals generated under the Neural Contractive Dynamical Systems (NCDS) setting, along with baseline comparisons using Multilayer Perceptron (MLP) and Neural Ordinary Differential Equation (NeuralODE) models.

Stability is commonly ensured through *asymptotic* or *contraction* guarantees. Asymptotic stability ensures that all motions converge to a fixed *point* (known as the system's *attractor*) (Khansari-Zadeh et al., 2011a, Khansari-Zadeh et al., 2014, Neumann et al., 2015, Zhang et al., 2022b, Zhang et al., 2022a). This is suitable when the only requirement is that the robot eventually takes a certain configuration, e.g. its end-effector is at a specific goal position. Many tasks, however, require the robot to dynamically follow desired trajectories, e.g. in flexible manufacturing, human-robot interaction, or in entertainment settings. In these cases, asymptotic stability is insufficient. A stronger notion of stability is provided by *contraction theory* (Lohmiller et al., 1998), which ensures that all the path integrals regardless of their initial state incrementally converge over time. Unfortunately, the mathematical requirements of a contractive system are difficult to ensure in popular neural network architectures.

Existing contractive learning methods focus on low-capacity models fitted under contraction constraints. Ravichandar et al. (2017) and Ravichandar et al. (2019) both use Gaussian mixture models to encode the demonstrations and model the dynamical system with Gaussian mixture regression (GMR), which is fitted under contraction

constraints. Blocher et al. (2017) also use GMR to model the system dynamics but instead include a stabilizing control input that guarantees a local contractive behavior. Likewise, Sindhwani et al. (2018) model the dynamical system with a well-crafted kernel, under which optimization remains convex w.r.t. contraction constraints. Note that the above works impose the contraction constraints during optimization, which complicates model training.

Developing high-capacity (deep) models is difficult as off-the-shelf neural network architectures provide no stability guarantees. Instead, current approaches split the task into two steps: first, learn a non-contractive dynamical system, and then estimate a Riemannian metric, parametrized by a neural network, under which the system becomes contractive (Tsukamoto et al., 2021d, Dawson et al., 2023). The approach, thus, requires training two separate networks, which is impractical and comes with additional challenges. For example, Sun et al. (2020) learns the metric by regularizing towards a contractive system, such that stability can *only* be ensured near training data. Alternatively, Tsukamoto et al. (2021a) develop a convex optimization problem for learning the metric from sampled *optimal* metrics. Unfortunately, accessing such optimal metrics is a difficult problem in itself.

In this thesis, we propose the *neural contractive dynamical system (NCDS)* which is the first neural network architecture that is guaranteed to be contractive for all parameter values (Beik-Mohammadi et al., 2024). The NCDS architecture is designed to be contractive for all parameter values, capable of being trained using standard neural network optimization techniques.

Hypothesis 3. *Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be a time-invariant vector field representing a dynamical system, parameterized by $\theta \in \Theta$. Suppose the Jacobian $\boldsymbol{J}_{f(x)}$ meets the necessary contraction condition (e.g., maintaining a strictly negative matrix measure) for all $\boldsymbol{x} \in \mathbb{R}^n$ and $\theta \in \Theta$. We hypothesize that constructing f as a neural network with built-in Jacobian constraints ensures that the resulting dynamical system is contractive for all parameter values.*

High-dimensional dynamical systems arise naturally in many real-world scenarios, where the state of a system is described by a large number of interdependent variables. Learning and understanding the behavior of such systems is essential, yet it presents significant challenges. In particular, ensuring desirable properties like contraction which promotes stability and robustness is especially difficult in high-dimensional settings due to the curse of dimensionality, which amplifies both computational and analytical complexity.

To address this, we propose a different route: instead of working directly in the high-dimensional space, we learn a contractive dynamical system in a low-dimensional latent space, where enforcing such properties is tractable. Then, we use an injective mapping to pull back the dynamics back into the original space. This mapping ensures that the contraction properties are preserved in the high-dimensional system. Furthermore, the approach is extended to cover full-pose end-effector movements, including orientation dynamics on the Lie groups ($\mathcal{S}^3$ and $\mathcal{SO}(3)$).

Hypothesis 4. *Let $g : \mathbb{R}^m \to \mathbb{R}^n$ be an injective function mapping high-dimensional demonstration data into the state space where f operates. We hypothesize that combining f with g preserves the contraction property across both translational and orientation dynamics, allowing the system to handle full-pose end-effector movements on the Lie groups $\mathcal{S}^3$ and $\mathcal{SO}(3)$ while remaining stable and convergent.*

1.3 Riemannian Manifolds for Safety Regions

Ensuring safe behavior is a fundamental requirement in autonomous systems, particularly in dynamic environments where unmodeled disturbances, sensor noise, and unseen obstacles pose significant risks. A key strategy to address this is the definition of *safety regions* – regions in the state space where the system is guaranteed to behave reliably, either by avoiding hazards or maintaining desirable stability properties.

Traditionally, safety regions are defined through constraint-aware control schemes such as Control Barrier Functions (CBFs) (Dawson et al., 2023). These methods, however, often rely on hand-crafted models or conservative approximations, which may not scale well to high-dimensional, data-driven settings.

In contrast, this thesis introduces a novel perspective rooted in the geometry of learned latent spaces. Specifically, it turns out that the *skill manifold* – a low-dimensional Riemannian manifold induced by a Variational Autoencoder (VAE) – encodes a measure of local uncertainty through the geometry of its pullback metric. This structure can be exploited to define *safety regions*.

We propose that the latent manifold, together with its associated pullback metric, enables the construction of modulation matrices that reshape the dynamical system locally. These modulations serve two purposes: they reshape the vector field in real time to steer the trajectory around dynamic obstacles, and they deflect the vector field in regions beyond the data support, effectively shrinking the region of confident operation

to prevent unsafe behaviors. This results in uncertainty-aware, robust control without sacrificing responsiveness.

This thesis demonstrates that the dynamical system generated by Neural Contraction Dynamical Systems (NCDS) can be locally reshaped using these modulation matrices while preserving contraction properties. These modulations are derived from Riemannian metrics defined over the latent space of the VAE (Beik-Mohammadi et al., 2025).

Hypothesis 5. *Consider a contractive dynamical system $f : \mathbb{R}^n \to \mathbb{R}^n$. Let $\mathcal{M}$ be a Riemannian manifold embedded in a latent space, from which we derive modulation matrices to locally reshape f. We hypothesize that these localized reshaping operations preserve the contractive property of f while enabling real-time obstacle avoidance in dynamic scenarios. Moreover, by leveraging the geometry of $\mathcal{M}$ to detect high-uncertainty regions, selectively modulating the vector field outside the data support maintains robustness and safety even in unobserved regions of the state space.*

1.4 Contributions

Here, we briefly summarize the key contributions.

1.4.1 Geodesic Motion Skills

We introduce a novel LfD method called *geodesic motion skills* (Beik-Mohammadi et al., 2021), which **constructs a Riemannian manifold from human demonstrations** in the task space of the robot. This method computes geodesics to accurately recover and synthesize motion skills (see Chapter 3).

- We develop a technique for **on-the-fly obstacle avoidance** by dynamically reshaping the Riemannian manifold using an ambient space metric (see Sec. 3.5.3).
- We propose a graph-based geodesic computation method for **real-time motion generation** (see Sec. 3.5.4).

1.4.2 Extension to Joint Space

We extend *geodesic motion skills* to **learn Riemannian manifolds in the joint space of the robot** (Beik-Mohammadi et al., 2023), enabling **multiple-limb obstacle avoidance** in real-time (see Chapter 4).

1.4.3 Neural Contractive Dynamical Systems (NCDS)

We propose the **first design mechanism for contractive neural networks** (Beik-Mohammadi et al., 2024, Beik-Mohammadi et al., 2025), introducing *neural contractive dynamical systems (NCDS)* to learn stable dynamical systems from demonstrations (see Chapter 5).

- We develop an injective variant of VAEs to guarantee **latent contractive systems decode to contractive dynamics**, addressing high-dimensional settings (see Chapter 6).
- We extend NCDS to **Lie groups** such as $\mathcal{SO}(3)$ and $\mathcal{S}^3$, to model complex robotic motions (see Sec. 6.2.2).
- We further extend NCDS to handle **conditional inputs**, enabling multi-modal data integration, including images (see Sec. 5.5).

1.4.4 Riemannian Implicit Distance Functions

We propose Riemannian implicit distance functions that exploit the intrinsic geometry of both learned data manifolds and obstacle regions via ambient space metrics (Beik-Mohammadi et al., 2025). This enables effective modulation of vector fields to achieve obstacle avoidance and avoid uncertain regions in contractive dynamical systems (see Sec. 5.4).

2 Background and Related Work

*This chapter is based on the paper **Learning Riemannian Manifolds for Geodesic Motion Skills**, published in Robotics: Science and Systems in 2021 (Beik-Mohammadi et al., 2021), **Reactive motion generation on learned Riemannian manifolds**, published in International Journal on Robotics Research (IJRR) in 2023 (Beik-Mohammadi et al., 2023), **Neural Contractive Dynamical Systems**, published in International Conference on Robot Learning (ICLR) in 2024 (Beik-Mohammadi et al., 2024), and the paper **Extended Neural Contractive Dynamical Systems: On Multiple Tasks and Riemannian Safety Regions**, submitted to International Journal on Robotics Research in 2024 (Beik-Mohammadi et al., 2025).*

Given the close alignment of this thesis with both trajectory-based techniques (Chapters 3 and 4) and dynamical systems (Chapter 5) within the framework of Learning from Demonstration (LfD), we discuss the relevant work in this chapter. Subsequently, we delve into the foundational concepts and key elements essential for a deeper understanding of this thesis.

2.1 Riemannian Geometry

Riemannian geometry serves as a foundational concept throughout this thesis, enabling the formulation of motion learning as a problem of understanding and leveraging the intrinsic geometry of demonstrated skills. In the trajectory-based approach, Riemannian manifolds provide a natural space in which geodesics – shortest paths under a learned metric – capture smooth and adaptable motions. In the dynamics-based approach, Lie groups offer a principled way to handle full-pose motions, particularly for orientation dynamics. Before delving into the specifics of learning motion, this section provides a thorough overview of Riemannian manifolds and Lie groups.

2.1.1 Riemannian Manifolds

In differential geometry, Riemannian manifolds are referred to as curved d-dimensional continuous and differentiable surfaces characterized by a Riemannian metric (Lee, 2018).

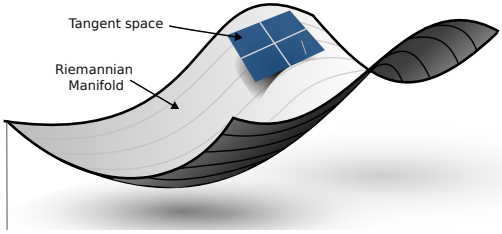


Figure 2.1: An illustration of a Riemannian manifold with a tangent space, highlighting the local Euclidean structure at a chosen point.

This metric is characterized by a family of smoothly varying positive-definite inner products acting on the tangent spaces of the manifold, which locally resembles the Euclidean space $\mathbb{R}^d$. The manifold and an example of its tangent space are illustrated in Fig. 2.1. In this thesis, we use the mapping function Ω to represent a manifold $\mathcal{M}$ immersed in the ambient space $\mathcal{X}$ defined as,

$$\mathcal{M} = \Omega(\mathcal{Z}) \quad \text{with} \quad \Omega : \mathcal{Z} \to \mathcal{X}, \tag{2.1}$$

where $\mathcal{Z}$ and $\mathcal{X}$ are open subsets of Euclidean spaces with $\dim \mathcal{Z} < \dim \mathcal{X}$.

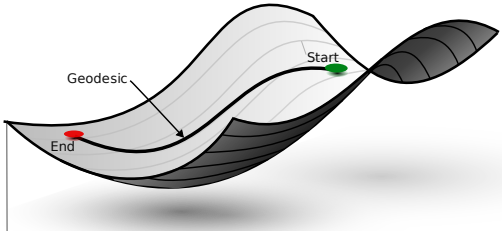


Figure 2.2: An illustration of a Riemannian manifold with a geodesic curve, representing the shortest path between two points.

An important operation on Riemannian manifolds is the computation of the length of a smooth curve $\zeta : [0,1] \to \mathcal{Z}$,

$$\mathcal{L}_\zeta = \int_0^1 \|\partial_t \Omega(\zeta(t))\| \mathrm{d}t. \tag{2.2}$$

This length can be reformulated using the chain rule as,

$$\mathcal{L}_\zeta = \int_0^1 \sqrt{\dot{\zeta}(t)^\mathsf{T} \boldsymbol{M}(\zeta(t)) \dot{\zeta}(t)} \mathrm{d}t, \tag{2.3}$$

where $\boldsymbol{M}$ and $\dot{\zeta}_t = \partial_t \zeta$ are the Riemannian metric and curve derivative, respectively. Note that the Riemannian metric corresponds to,

$$\boldsymbol{M}(\boldsymbol{z}) = \boldsymbol{J}_\Omega(\boldsymbol{z})^\mathsf{T} \boldsymbol{J}_\Omega(\boldsymbol{z}). \tag{2.4}$$

Here, $\boldsymbol{J}_\Omega(\boldsymbol{z})$ is the Jacobian of the mapping function Ω. This metric can be used to measure local distances in $\mathcal{Z}$. As illustrated in Fig. 2.2, the shortest path on the manifold, also known as the geodesic $\gamma(t)$, can be computed by minimizing the curve length in (2.3).

The geodesic curve $\gamma(t)$ can also be obtained by solving the following second-order differential equation, known as the *geodesic equation*:

$$\frac{d^2\gamma^i}{dt^2} + \Gamma^i_{jk}(\gamma(t)) \frac{d\gamma^j}{dt} \frac{d\gamma^k}{dt} = 0, \tag{2.5}$$

where Γ^i_{jk} are the Christoffel symbols of the Riemannian metric. This system of equations is generally nonlinear and difficult to solve analytically, especially in high-dimensional learned manifolds, and therefore requires numerical methods such as spline optimization or graph-based search (see Sec. 3.5)(Arvanitidis et al., 2018).

2.1.2 Lie Groups

Lie groups are mathematical structures that combine the properties of groups and differentiable manifolds, allowing for the study of continuous symmetries. They are used extensively in various fields, including physics, robotics, and computer vision, to model continuous transformations such as rotations, translations, and scaling. A Lie group is a Riemannian manifold equipped with a Lie group structure, where the group

operations—multiplication and inversion—are smooth maps. This smooth algebraic structure, combined with the manifold's geometry, makes Lie groups a powerful tool for applying differential geometry to study continuous symmetries and transformations.

Lie algebra

Lie algebras are mathematical structures that are closely associated with Lie groups. A Lie algebra provides a way to study the properties and behavior of a Lie group through a linearization of the group near the identity element. Essentially, a Lie algebra captures the infinitesimal behavior of a Lie group. To define a Lie algebra formally, consider a Lie group G. The corresponding Lie algebra, denoted as $\mathfrak{g}$, is the tangent space at the identity element of the group (e.g. the north pole of the hypersphere S^3). This tangent space can be thought of as the set of all possible directions in which one can move away from the identity element within the group.

Lie operators

The concept of Lie operators provides a mathematical framework for moving between a Lie group and its associated Lie algebra. For example, for the special orthogonal group, $\mathcal{SO}(3)$, which represents 3 dimensional rotations, and its corresponding Lie algebra, $\mathfrak{so}(3)$, we use two key mappings: the logarithmic map (Log) and the exponential map (Exp).

Exponential Map ($\text{Exp} : \mathfrak{g} \to G$**):** For certain Lie groups such as $\mathcal{SO}(3)$, the exponential map translates an element of the Lie algebra, represented as a skew-symmetric matrix, into an element of the Lie group, e.g., a rotation matrix. However, this correspondence does not universally apply to all Lie groups.

This mapping captures how infinitesimal rotations in the algebra are integrated into finite rotations. The result is a smooth way of generating rotation matrices from their algebraic representations.

Logarithmic Map ($\text{Log} : G \to \mathfrak{g}$**):** Conversely, the logarithmic map recovers a Lie algebra element from a group element in $\mathcal{SO}(3)$, typically yielding a skew-symmetric matrix representation. This allows us to analyze rotations in terms of their infinitesimal components. Both maps rely on the structure of the group and algebra, which involves mathematical tools such as the trace of a matrix (to determine the rotation angle) and the skew-symmetric matrix form (to represent rotation axes).

Later, we will define these operators explicitly for different Lie groups, illustrating how our framework adapts to various geometric structures.

2.2 Variational Auto-encoders (VAEs)

In robot motion learning, dealing with high-dimensional demonstration data directly is both computationally intensive and detrimental to effective learning and reconstruction. To overcome this obstacle, we first convert this data into a compact, low-dimensional representation that retains the essential features necessary for motion synthesis. This transformation not only eases the computational burden but also provides a critical foundation for calculating Riemannian manifolds. These manifolds are then used to efficiently learn and generalize motion skills.

To achieve this, we employ generative modeling techniques that inherently facilitate the transition from high-dimensional observations to low-dimensional latent representations.

VAEs are generative models (Kingma et al., 2014b) that learn and reconstruct data by encapsulating their density into a lower-dimensional latent space $\mathcal{Z}$. Specifically, VAEs encode the training data density $p(\boldsymbol{x})$ in an ambient space $\mathcal{X}$ through a low-dimensional latent variable $\boldsymbol{z}$. For simplicity, we consider the generative process of a Gaussian VAE defined as,

$$p(\boldsymbol{z}) = \mathcal{N}\left(\boldsymbol{z}|\mathbf{0}, \mathbb{I}_d\right), \qquad \boldsymbol{z} \in \mathcal{Z}; \tag{2.6}$$

$$p_{\boldsymbol{\phi}}(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}\left(\boldsymbol{x}|\mu_{\boldsymbol{\phi}}(\boldsymbol{z}), \mathbb{I}_D\sigma^2_{\boldsymbol{\phi}}(\boldsymbol{z})\right), \qquad \boldsymbol{x} \in \mathcal{X}. \tag{2.7}$$

where $\mu_{\boldsymbol{\phi}} : \mathcal{Z} \to \mathcal{X}$ and $\sigma_{\boldsymbol{\phi}} : \mathcal{Z} \to \mathbb{R}^D_+$ are deep neural networks with parameters $\boldsymbol{\phi}$ estimating the mean and the variance of the posterior distribution $p_{\boldsymbol{\phi}}(\boldsymbol{x}|\boldsymbol{z})$, and $\mathbb{I}_D$ and $\mathbb{I}_d$ are identity matrices of size D and d, respectively. Since the exact inference of the generative process is in general intractable, a variational approximation of the evidence (marginal likelihood) can be used,

$$\mathcal{L}_{ELBO}(\boldsymbol{x}) = \mathbb{E}_{q_{\xi}(\boldsymbol{z}|\boldsymbol{x})}\left[\log(p_{\boldsymbol{\phi}}(\boldsymbol{x}|\boldsymbol{z}))\right] - \mathrm{KL}\left(q_{\xi}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})\right), \tag{2.8}$$

where $q_{\xi}(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\mu_{\xi}(\boldsymbol{x}), \mathbb{I}_d\sigma^2_{\xi}(\boldsymbol{x}))$ approximates the posterior distribution $p(\boldsymbol{z}|\boldsymbol{x})$ by two deep neural networks $\mu_{\xi}(\boldsymbol{x}) : \mathcal{X} \to \mathcal{Z}$ and $\sigma_{\xi}(\boldsymbol{x})) : \mathcal{X} \to \mathbb{R}^d_+$. The posterior distribution $p_{\xi}(\boldsymbol{z}|\boldsymbol{x})$ is called *inference* or *encoder* distribution, while the generative

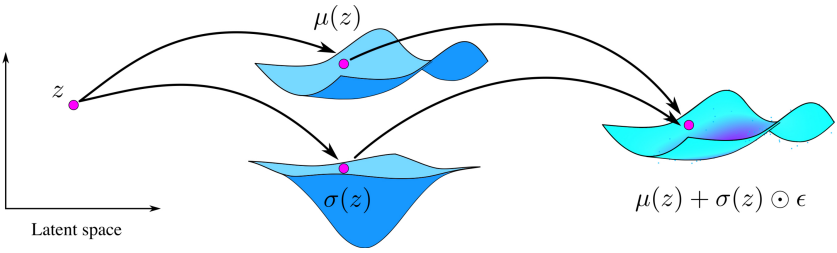


Figure 2.3: In a Gaussian VAE, samples are generated by a random projection of the manifold jointly spanned by μ and σ.

distribution $p_{\phi}(\boldsymbol{x}|\boldsymbol{z})$ is known as the *generator* or *decoder*. In the next subsection, we use a VAE to learn a skill-specific Riemannian manifold from human demonstrations.

2.2.1 Injective Flows

While geodesic motion generation operates purely on the latent manifold and does not require the decoder to be injective, learning stable dynamical systems in latent space introduces a stronger requirement. In particular, to transfer vector fields between the latent and ambient spaces while preserving stability properties, the decoder must be injective. This ensures that each point in the latent space corresponds to a unique point in the ambient space, allowing the learned dynamics to be meaningfully and consistently mapped back to the robot's full state space. Injective flows provide a principled way to construct such mappings, enabling the use of latent-space stability while ensuring the resulting dynamics remain stable in the high-dimensional observation space. A limitation of VAEs is that their marginal likelihood is intractable and we have to rely on a bound. When $\dim(\mathcal{X}) = \dim(\mathcal{Z})$, we can apply the change-of-variables theorem to evaluate the marginal likelihood exactly, giving rise to *normalizing flows* (Tabak et al., 2013). This requires the decoder to be diffeomorphic, i.e. a smooth invertible function with a smooth inverse. In order to extend this to the case where $\dim(\mathcal{X}) > \dim(\mathcal{Z})$, Brehmer et al. (2020) proposed an *injective flow*, which implements a zero-padding operation (see Sec.6.1 and Equation 6.1) on the latent variables alongside a diffeomorphic decoder, such that the resulting function is injective. Later, in Sec. 6.1 , we formalize how the injectivity of this mapping ensures the learnability of stable dynamical systems within low-dimensional latent spaces.

2.3 Learning Riemannian Manifolds with VAEs

We here explain the link between VAEs and Riemannian geometry. It should be mentioned that learning Riemannian manifolds with a VAE is independent of the injectivity of the decoder. To begin, we define the VAE generative process of (2.7) as a stochastic function,

$$f_{\boldsymbol{\phi}}(\boldsymbol{z}) = \mu_{\boldsymbol{\phi}}(\boldsymbol{z}) + \text{diag}(\epsilon)\sigma_{\boldsymbol{\phi}}(\boldsymbol{z}), \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbb{I}_D), \tag{2.9}$$

where $\mu_{\boldsymbol{\phi}}(\boldsymbol{z})$ and $\sigma_{\boldsymbol{\phi}}(\boldsymbol{z})$ are decoder mean and variance neural networks, respectively. Also, $\text{diag}(\cdot)$ is a diagonal matrix, and $\mathbb{I}_D$ is a $D \times D$ identity matrix. The above formulation is referred to as the reparameterization trick (Kingma et al., 2014b), which can be interpreted as samples generated out of a random projection of a manifold jointly spanned by $\mu_{\boldsymbol{\phi}}$ and $\sigma_{\boldsymbol{\phi}}$ (Eklund et al., 2019).

Riemannian manifolds may arise from mapping functions between two spaces as in (2.1). As a result, (2.9) may be seen as a stochastic version of the mapping function of (2.1), which in turn defines a Riemannian manifold (Hauberg, 2019). We can now write the stochastic form of the Riemannian metric of (2.4). To do so, we first recast the stochastic function (2.9) as follows (Eklund et al., 2019),

$$f_{\boldsymbol{\phi}}(\boldsymbol{z}) = \begin{pmatrix} \mathbb{I}_D, & \text{diag}(\epsilon) \end{pmatrix} \begin{pmatrix} \mu_{\boldsymbol{\phi}}(\boldsymbol{z}) \\ \sigma_{\boldsymbol{\phi}}(\boldsymbol{z}) \end{pmatrix} = \boldsymbol{P}\ s(\boldsymbol{z}), \tag{2.10}$$

where $\boldsymbol{P}$ is a random matrix, and $s(\boldsymbol{z})$ is the concatenation of $\mu_{\boldsymbol{\phi}}(\boldsymbol{z})$ and $\sigma_{\boldsymbol{\phi}}(\boldsymbol{z})$. Therefore, the VAE can be seen as a random projection of a deterministic manifold spanned by s. Given that this stochastic mapping function is defined by a combination of mean $\mu_{\boldsymbol{\phi}}(\boldsymbol{z})$ and variance $\sigma_{\boldsymbol{\phi}}(\boldsymbol{z})$, the expected Riemannian metric is likewise based on a mixture of both as follows,

$$\boldsymbol{M}(\boldsymbol{z}) = \boldsymbol{J}_{\mu_{\boldsymbol{\phi}}}(\boldsymbol{z})^{\mathsf{T}} \boldsymbol{J}_{\mu_{\boldsymbol{\phi}}}(\boldsymbol{z}) + \boldsymbol{J}_{\sigma_{\boldsymbol{\phi}}}(\boldsymbol{z})^{\mathsf{T}} \boldsymbol{J}_{\sigma_{\boldsymbol{\phi}}}(\boldsymbol{z}), \tag{2.11}$$

where $\boldsymbol{J}_{\mu_{\boldsymbol{\phi}}}(\boldsymbol{z})$ and $\boldsymbol{J}_{\sigma_{\boldsymbol{\phi}}}(\boldsymbol{z})$ are respectively the Jacobian of $\mu_{\boldsymbol{\phi}}(\boldsymbol{z})$ and $\sigma_{\boldsymbol{\phi}}(\boldsymbol{z})$ evaluated at $\boldsymbol{z} \in \mathcal{Z}$, with $\mathcal{Z}$ being the VAE low-dimensional latent space. Note that we assume the noise ϵ is sampled from a normal distribution with zero mean and unit covariance, thus the contribution from the variance term vanishes in expectation. With access to this metric, we can locally reshape the manifold by modifying the metric in targeted regions. For example, we can increase the local metric volume in areas where obstacles are present – effectively increasing the local curvature (i.e., the degree of metric distortion that governs how geodesics bend). This adjustment makes paths through these regions

less favorable. Next, we discuss how this adjusted curvature can be transformed into a metric that aids in obstacle avoidance within the latent space.

2.4 Ambient Space Metric

In the previous sections, we described how a VAE can be used to learn a Riemannian metric over a latent space by interpreting the decoder as a smooth mapping from the latent to the ambient space. This metric captures the intrinsic geometric structure of the demonstrations and allows for the computation of geodesics that reconstruct motion patterns.

However, in practical applications such as robot motion generation, it is often necessary to modify the learned metric to accommodate additional requirements, such as obstacle avoidance. In such cases, the original metric – although well-suited to encode the main patterns of the demonstrations – may not account for external constraints or safety considerations. To address this, we introduce an ambient space metric that enables us to reshape the latent geometry in a way that reflects these task-specific demands. This means that we now require to reshape the manifold to take these obstacles into account. To do so, we need to reshape the previously-learned metric so that the new geodesics lead to obstacle-free robot motions.

A naïve approach would entail retraining the VAE model with new data, which is time-consuming and data-intensive, and can only be executed offline. We propose to reshape the learned metric by considering problem-specific ambient metrics. Note that although the definition of curve length relies on the Euclidean metric of $\mathcal{X}$, this is not a strict requirement. Indeed, Arvanitidis et al. (2021) argued that there is value in giving the ambient space a manually-defined *Riemannian* metric and including that into the definition of curve length. The resulting metric can then be used to compute the curve length as,

$$\mathcal{L}_c = \int_0^1 \sqrt{\dot{c}(t)^\top \boldsymbol{J}_{f_{\boldsymbol{\phi}}}(c(t))^\top \boldsymbol{M}_{\mathcal{X}}(f_{\boldsymbol{\phi}}(c(t))) \boldsymbol{J}_{f_{\boldsymbol{\phi}}}(c(t)) \dot{c}(t)} \mathrm{d}t, \tag{2.12}$$

where $\boldsymbol{M}_{\mathcal{X}}$ is the ambient metric, which can now vary smoothly across $\mathcal{X}$. The reshaped Riemannian metric in $\mathcal{Z}$ is then computed as follows,

$$\bar{\boldsymbol{M}}(\boldsymbol{z}) = \boldsymbol{J}_{\mu_{\boldsymbol{\phi}}}(\boldsymbol{z})^\top \boldsymbol{M}_{\mathcal{X}}(\mu_{\boldsymbol{\phi}}(\boldsymbol{z})) \boldsymbol{J}_{\mu_{\boldsymbol{\phi}}}(\boldsymbol{z}) + \boldsymbol{J}_{\sigma_{\boldsymbol{\phi}}}(\boldsymbol{z})^\top \boldsymbol{M}_{\mathcal{X}}(\mu_{\boldsymbol{\phi}}(\boldsymbol{z})) \boldsymbol{J}_{\sigma_{\boldsymbol{\phi}}}(\boldsymbol{z}). \tag{2.13}$$

Given this metric, we can compute geodesics that are repelled from certain regions of the ambient space $\mathcal{X}$ by increasing the value of the ambient metric $\boldsymbol{M}_{\mathcal{X}}$. We demonstrate how this metric reshaping method is leveraged to generate obstacle-free robot motions both in trajectory-based motion skills and dynamical systems.

2.5 Learning from Demonstration (LfD)

LfD is a robot programming technique that leverages human demonstrations, recorded via kinesthetic teaching or teleoperation, to learn a model of a task (Ravichandar et al., 2020). Examples of data used in LfD include end-effector positions and orientations, joint configurations, linear or angular velocities, and accelerations. Unlike traditional motion planning algorithms, which rely on predefined mathematical models and constraints to compute feasible paths, LfD directly derives task-relevant behaviors from demonstration data.

We identify three core frameworks in robot motion learning, each characterized by specific features such as obstacle avoidance. The first framework employs probabilistic methods to exploit data variability and model uncertainty (Huang et al., 2019, Calinon, 2016, Paraschos et al., 2018). Their probabilistic formulation allows robots to execute skills using diverse trajectories sampled from the skill model. The second framework includes neural network approaches for enhancing generalization in robot motion learning (Seker et al., 2019, Osa et al., 2019). All these methods fall under the movement primitives (MPs) category, offering an alternative to traditional motion planning (Elbanhawi et al., 2014) for robot motion generation. The last framework focuses on motion dynamics learning, where demonstrations are considered solutions to specific dynamical systems (Ijspeert et al., 2013, Manschitz et al., 2018). Additionally, hybrid methods combining dynamical systems and neural networks (Bahl et al., 2020, Rana et al., 2020a), or dynamical systems and probabilistic models (Ugur et al., 2020, Khansari-Zadeh et al., 2011b), have also been proposed. Focusing on the first two categories (trajectory-based approaches), in Chapters 3 and 4, we propose a reactive motion generation technique positioned between movement primitives and motion planning. Like motion primitives, our approaches use human demonstrations to learn a model of a skill, assuming that these demonstrations form a smooth surface characterized as a Riemannian manifold. Simultaneously, akin to motion planners, our method derives a time-independent reference trajectory generated from geodesic curves, which can be locally deformed to avoid unseen obstacles.

Our approach utilizes a neural network, specifically a variational autoencoder (VAE) (Kingma et al., 2014b), to learn a Riemannian metric that incorporates the network's uncertainty. This metric facilitates generating motions resembling the demonstrations through geodesics. The decoded geodesics then serve as reference trajectories to reproduce motion trajectories similar to the demonstrations.

Sophisticated robot movements often involve full-pose end-effector trajectories, necessitating a framework capable of encoding such motion patterns. While many LfD approaches overlook this challenge (Paraschos et al., 2018, Seker et al., 2019, Calinon, 2016, Huang et al., 2019), recent efforts address it using probabilistic models (Rozo et al., 2021, Zeestraten, 2018).

Obstacle avoidance is another critical feature of reactive motion generation. Many approaches rely on predefined obstacle information (Prescott et al., 1992, Aljalbout et al., 2020), which proves ineffective in dynamic environments. Some techniques indirectly address this through via-points (Paraschos et al., 2018, Seker et al., 2019, Huang et al., 2019) without retraining the skill model. Other methods treat obstacles as costs in an optimization framework to generate optimal, obstacle-free trajectories (Urain et al., 2021, Ratliff et al., 2018).

Our method tackles obstacle avoidance through metric reshaping. We design obstacle-aware ambient space metrics to reshape the learned Riemannian metric. The combination of these metrics generates trajectories that follow the demonstrations while avoiding obstacles. Notably, the choice of demonstration ambient space influences the metric design. For instance, joint space demonstrations allow incorporating information about distances from any robot point to obstacles, enabling multiple-limb obstacle avoidance. Conversely, task space end-effector demonstrations restrict obstacle avoidance to the robot's end-effector level.

Our obstacle avoidance technique, introduced in Chapters 3 and 4, resembles CHOMP (Ratliff et al., 2009), employing a simplified geometric representation of the robot and obstacles to construct a uniform grid in the task space. This grid identifies potential collisions, and the information is integrated into geodesic energy, leading to optimal paths. Like CHOMP, this approach leverages a cost function incorporating obstacles.

Moreover, human demonstrations often exhibit multiple solutions to a single motion skill (Rozo et al., 2011, Seker et al., 2019), typically handled using hierarchical methods (Konidaris et al., 2012, Ewerton et al., 2015). As detailed in Chapters 3 and 4, our approach encodes multiple-solution tasks on the learned Riemannian manifold,

enabling replication of demonstrated skills and generating novel hybrid solutions. These hybrid solutions emerge naturally from our geodesic motion generator on the Riemannian manifold. In contrast, existing techniques generally restrict trajectories to demonstrated patterns without supporting hybrid solutions.

Concerning the third category built on dynamical system representations, unlike previous approaches, dynamical systems explicitly consider the time aspect of motion, making them well-suited for modeling time-evolving behaviors. For dynamical systems, stability is a crucial property as it ensures predictable and consistent behavior over time, which is essential for robust motion generation. Stability guarantee methods are divided into two categories. The first involves systems demonstrating stability towards a single attractor, referred to as asymptotic stability, where the system eventually settles at a predetermined point, regardless of its initial state (Giesl et al., 2015). The second includes contractive dynamical systems, which describe systems where initially close states grow closer over time (Jouffroy et al., 2010, Tsukamoto et al., 2021e). This characteristic ensures high stability, enabling accurate trajectory following.

While techniques like normalizing flows in the first category create motions resembling a trajectory, they do not guarantee exact reconstruction, only reaching the fixed point (Urain et al., 2020, Rana et al., 2020a, Zhang et al., 2022a). Next, we focus on contraction theory principles and methods for learning these systems. Additionally, we overview injective flows and modulation matrices relevant to the proposed approach in this thesis.

2.6 Contraction in Dynamical Systems

Assume an autonomous dynamical system $\dot{\boldsymbol{x}}_t = f(\boldsymbol{x}_t)$, where $\boldsymbol{x}_t \in \mathbb{R}^D$ is the state variable, $f : \mathbb{R}^D \to \mathbb{R}^D$ is a $\boldsymbol{C}^1$ function, and $\dot{\boldsymbol{x}}_t = {}^{\mathrm{d}x}/_{\mathrm{d}t}$ denotes temporal differentiation. As depicted in Fig. 5.1, contraction stability guarantees that all solution trajectories of a nonlinear system f incrementally converge regardless of initial conditions $\boldsymbol{x}_0$,$\dot{\boldsymbol{x}}_0$, and temporary perturbations (Lohmiller et al., 1998). The system stability can, thus, be analyzed differentially, i.e. we can ask if two nearby trajectories converge to one another. Specifically, contraction theory defines a measure of distance between neighboring trajectories, known as the *contraction metric*, under which the distance decreases exponentially over time (Jouffroy et al., 2010, Tsukamoto et al., 2021e).

Formally, an autonomous dynamical system yields the differential relation $\delta\dot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{x})\delta\boldsymbol{x}$, where $\boldsymbol{J}(\boldsymbol{x}) = {}^{\partial f}/_{\partial \boldsymbol{x}}$ is the system Jacobian and $\delta\boldsymbol{x}$ is a virtual displacement (i.e., an infinitesimal spatial displacement between the nearby trajectories at a fixed time).

Note that we have dropped the time index t to limit notational clutter. The rate of change of the corresponding infinitesimal squared distance $\delta\boldsymbol{x}^\top\delta\boldsymbol{x}$ is

$$\frac{\mathrm{d}}{\mathrm{d}t}(\delta\boldsymbol{x}^\top\delta\boldsymbol{x}) = 2\delta\boldsymbol{x}^\top\delta\dot{\boldsymbol{x}} = 2\delta\boldsymbol{x}^\top\boldsymbol{J}(\boldsymbol{x})\delta\boldsymbol{x}. \tag{2.14}$$

It follows that if the symmetric part of the Jacobian $\boldsymbol{J}(\boldsymbol{x})$ is negative definite, then the infinitesimal squared distance $\delta\boldsymbol{x}^\top\delta\boldsymbol{x}$ between neighboring trajectories decreases over time. This is formalized as follows.

Definition 2.1 (Contraction stability (Lohmiller et al., 1998))**.** *An autonomous dynamical system $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$ exhibits a contractive behavior if its Jacobian $\boldsymbol{J}(\boldsymbol{x}) = {}^{\partial f}/_{\partial \boldsymbol{x}}$ is uniformly negative definite, or equivalently if its symmetric part is negative definite. This means that there exists a constant $\tau > 0$ such that $\delta\boldsymbol{x}^\top\delta\boldsymbol{x}$ converges to zero exponentially at rate 2τ, i.e., $\|\delta\boldsymbol{x}\| \leq e^{-\tau t}\|\delta\boldsymbol{x}_0\|$. This can be summarized as,*

$$\exists\, \tau > 0 \;\; s.t. \;\; \forall \boldsymbol{x}, \;\; \frac{1}{2}\Big(\boldsymbol{J}(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})^\top\Big) \prec -\tau\mathbb{I} \prec 0. \tag{2.15}$$

The above analysis can be generalized to account for a more general notion of distance of the form $\delta\boldsymbol{x}^\top\boldsymbol{M}(\boldsymbol{x})\delta\boldsymbol{x}$, where $\boldsymbol{M}(\boldsymbol{x})$ is a positive-definite matrix known as the Riemannian contraction metric (Tsukamoto et al., 2021e, Dawson et al., 2023). In Chapter 5, we learn a dynamical system f so that it is inherently contractive since its Jacobian $\boldsymbol{J}(\boldsymbol{x})$ fulfills the condition given in (2.15). Consequently, it is not necessary to separately learn a contraction metric as an identity matrix suffices.

2.7 Contraction-preserving Obstacle Avoidance via Matrix Modulation

In this section, we review the matrix modulation technique for obstacle avoidance (Huber et al., 2019). Subsequently, in Sec. 6.3, we will explore how this method can be extended using Riemannian manifold learning to navigate obstacles and avoid unsafe regions. In a dynamic environment with obstacles, a learned contractive dynamical system should effectively adapt to and avoid unseen obstacles, without interfering with the

global contracting behavior of the system. This approach locally reshapes the learned vector field in the proximity of obstacles, while preserving contraction properties.

Specifically, Huber et al. (2022) show how to construct a modulation matrix $\boldsymbol{G}$ from the obstacle's location and geometry, such that the vector field $\dot{\boldsymbol{x}} = \boldsymbol{G}(\boldsymbol{x}) f_\theta(\boldsymbol{x})$ is both contractive and steers around the obstacle. Formally, given the modulation matrix $\boldsymbol{G}$, we can reshape the vector field

$$\hat{\dot{\boldsymbol{x}}} = \boldsymbol{G}(\boldsymbol{x}) f_\theta(\boldsymbol{x}), \tag{2.16}$$

$$\boldsymbol{G}(\boldsymbol{x}) = \boldsymbol{E}(\boldsymbol{x}) \boldsymbol{D}(\boldsymbol{x}) \boldsymbol{E}(\boldsymbol{x})^{-1}, \tag{2.17}$$

where $\boldsymbol{E}(\boldsymbol{x})$ and $\boldsymbol{D}(\boldsymbol{x})$ are the basis and diagonal eigenvalue matrices computed as,

$$\boldsymbol{E}(\boldsymbol{x}) = [\boldsymbol{n}(\boldsymbol{x})\ \mathbf{e}_1(\boldsymbol{x}) \dots \mathbf{e}_{d-1}(\boldsymbol{x})], \tag{2.18}$$

$$\boldsymbol{D}(\boldsymbol{x}) = \mathrm{diag}(\lambda_n(\boldsymbol{x})\lambda_\tau(\boldsymbol{x}), \dots, \lambda_\tau(\boldsymbol{x})), \tag{2.19}$$

where $\boldsymbol{n}(\boldsymbol{x}) = \frac{\boldsymbol{x}-\boldsymbol{x}_r}{\|\boldsymbol{x}-\boldsymbol{x}_r\|}$ is a reference direction computed w.r.t. a reference point $\boldsymbol{x}_r$ on the obstacle, and the tangent vectors $\mathbf{e}_i$ form an orthonormal basis to the gradient of the distance function $\Gamma(\boldsymbol{x})$ (see Huber et al. (2022) for its full derivation). Moreover, the components of the matrix $\boldsymbol{D}$ are defined as $\lambda_n(\boldsymbol{x}) = 1 - \left(\frac{1}{\Gamma(\boldsymbol{x})}\right)^{\frac{1}{\rho}}$, $\lambda_\tau(\boldsymbol{x}) = 1 + \left(\frac{1}{\Gamma(\boldsymbol{x})}\right)^{\frac{1}{\rho}}$, where $\rho \in \mathbb{R}^+$ is a reactivity factor. Note that the matrix $\boldsymbol{D}$ modulates the dynamics along the directions of the basis defined by the set of vectors $\boldsymbol{n}(\boldsymbol{x})$ and $\mathbf{e}(\boldsymbol{x})$. As stated by Huber et al. (2022), the function $\Gamma(\cdot)$ monotonically increases w.r.t the distance from the obstacle's reference point $\boldsymbol{x}_r$, and it is, at least, a C^1 function. Importantly, the modulated dynamical system $\hat{\dot{\boldsymbol{x}}} = \boldsymbol{G}(\boldsymbol{x}) f_\theta(\boldsymbol{x})$ still guarantees contractive stability, which can be proved by following the same proof provided by Huber et al. (2019).

3 Geodesic Motion Skills in Task Space

*This chapter is based on the paper **Learning Riemannian Manifolds for Geodesic Motion Skills**, published in Robotics: Science and Systems in 2021 (Beik-Mohammadi et al., 2021).*

In this chapter, we describe how learning complex robot motion skills from demonstrations can be treated from a Riemannian manifold perspective. Unlike previous works (Havoutis et al., 2013, Li et al., 2018), where skill manifolds are built from locally smooth manifold learning (Dollár et al., 2007), we leverage a Riemannian formulation. We develop a model that has enough capacity to learn and synthesize the relevant patterns of a motion while being flexible enough to adapt to new conditions (e.g. dynamic obstacles). An example is shown in Fig.1.1–*left*, where a Riemannian manifold is learned from demonstrations of a circular motion. The geodesics on this manifold naturally reproduce the shape of the skill, illustrating how the model captures the underlying geometry of the task.

In this context, related approaches such as geometric control methods build on the geometric properties of a system in the design of control laws. These methods are particularly useful in situations where the dynamics of the system are complex and difficult to model accurately (Bullo et al., 2004).

On a related note, our approach shares some conceptual aspects with optimal control theory. Broadly speaking, optimal control aims at finding optimal control inputs that minimize a specific cost functional (Kirk, 1970). The connection between inverse optimal control and Riemannian manifold learning is that the learned Riemannian metric can be understood as the cost function that the human demonstrator is optimizing. More specifically, the optimal expert demonstrations of a motion are used to learn a Riemannian metric that defines the optimal or shortest path (i.e., geodesics) that minimizes the functional in (2.3). In comparison, our method differs conceptually from the aforementioned control techniques, as it leverages geodesic on a learned Riemannian manifold for robot motion generation. While all approaches aim to achieve precise and efficient control or planning for mechanical systems, they do so through

different perspectives. We next describe how we use VAEs to access a low-dimensional learned manifold of the demonstrations to learn an ambient space Riemannian metric.

As mentioned previously, this metric is exploited to reconstruct robot motions in both task space $\mathbb{R}^3 \times \mathcal{S}^3$ and joint space $\mathbb{R}^n$ (described in Chapter 4). In addition, we discuss the design of the corresponding VAE for each ambient space as well as the formulation of the corresponding Riemannian metric. In Section 3.5, we explain how we exploit these learned metrics to generate robot motion trajectories using geodesics.

To begin, we focus on learning motion skills characterized by full-pose end-effector trajectories, where each pose is represented in $\mathbb{R}^3 \times \mathcal{S}^3$. Before exploiting the VAE to compute the Riemannian metric, we must ensure its capability to properly learn and reconstruct full-pose end-effector states, i.e. position $\boldsymbol{x} \in \mathbb{R}^3$ and orientation $\boldsymbol{q} \in \mathcal{S}^3$, while accounting for specific properties of the data, such as quaternions antipodality. To do so, we propose a VAE architecture that models the joint density of the robot end-effector state. Our model retains the usual Gaussian prior $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{0}, \mathbb{I}_d)$, but modifies the generative distribution $p_{\boldsymbol{\phi},\boldsymbol{\psi}}(\boldsymbol{x}, \boldsymbol{q}|\boldsymbol{z})$. Specifically, we assume that position and orientation are conditionally independent,

$$p_{\boldsymbol{\phi},\boldsymbol{\psi}}(\boldsymbol{x}, \boldsymbol{q}|\boldsymbol{z}) = p_{\boldsymbol{\phi}}(\boldsymbol{x}|\boldsymbol{z}) p_{\boldsymbol{\psi}}(\boldsymbol{q}|\boldsymbol{z}), \tag{3.1}$$

where the latent variable $\boldsymbol{z}$ captures the correlation between position and quaternion data. Next, we describe how each conditional distribution is parameterized and learned.

3.1 Position Encoding

To model the conditional distribution of end-effector positions $\boldsymbol{x}$, we opt for simplicity and choose this to be Gaussian,

$$p_{\boldsymbol{\phi}}(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}|\mu_{\boldsymbol{\phi}}(\boldsymbol{z}), \mathbb{I}_3 \sigma^2_{\boldsymbol{\phi}}(\boldsymbol{z})), \tag{3.2}$$

where $\mu_{\boldsymbol{\phi}}$ and $\sigma_{\boldsymbol{\phi}}$ are neural networks parametrized by $\boldsymbol{\phi}$.

This is a somewhat simplistic model as the Gaussian model will assign probability mass outside the workspace of the robot, i.e. physically infeasible robot states. With simplicity in mind, we disregard this concern as we only optimize the associated likelihood, and therefore never sample infeasible states. However, if we were to use the VAE as a fully generative model, this likelihood should be replaced by a more elaborated model that accounts for physical properties.

3.2 Quaternion Encoding

On a robot motion trajectory, each position is paired with an orientation, and together they define the full pose of the end-effector. There are several representations for the end-effector orientation, for example, Euler angles, rotation matrices, and unit quaternions. Euler angles and rotation matrices are widely used for their simplicity and intuitiveness, however, the former suffer from gimbal lock (Hemingway et al., 2018) which makes them an inadequate orientation parametrization, and the latter are a redundant representation requiring a high number of parameters.

Unit quaternions, on the other hand, are a convenient way to represent orientations since they are compact, not redundant, and prevent gimbal locks. Also, they provide strong stability guarantees in closed-loop orientation control (Campa et al., 2008), and they have been recently exploited in complicated robotic tasks learning (Rozo et al., 2020), and for data-efficient robot control tuning (Jaquier et al., 2020) using Riemannian-manifold formulations. We choose to represent orientations $\boldsymbol{q}$ as a unit quaternion, such that $\boldsymbol{q} \in \mathcal{S}^3$ with the additional antipodal identification that $\boldsymbol{q}$ and $-\boldsymbol{q}$ correspond to the same orientation. Formally, a unit quaternion $\boldsymbol{q}$ lying on the surface of a 3-sphere $\mathcal{S}^3$ can be represented using a 4-dimensional unit vector $\boldsymbol{q} = [q_w, q_x, q_y, q_z]$, where the scalar q_w and vector (q_x, q_y, q_z) represent the real and imaginary parts of the quaternion, respectively. To cope with antipodality, one could model $\boldsymbol{q}$ as a point in a projective space, but for simplicity we let $\boldsymbol{q}$ live on the unit sphere $\mathcal{S}^3$. We then choose a generative distribution $p_{\boldsymbol{\psi}}(\boldsymbol{q}|\boldsymbol{z})$ such that $p_{\boldsymbol{\psi}}(\boldsymbol{q}|\boldsymbol{z}) = p_{\boldsymbol{\psi}}(-\boldsymbol{q}|\boldsymbol{z})$. In other words, the quaternions $\boldsymbol{q}$ and $-\boldsymbol{q}$ are considered to be antipodal: they lie on diametrically opposite points on the 3-sphere while representing the same orientation.

To formulate a suitable distribution $p_{\boldsymbol{\psi}}(\boldsymbol{q}|\boldsymbol{z})$ over $\mathcal{S}^3$, we leverage the von Mises-Fisher (vMF) distribution, which is merely an isotropic Gaussian constrained to lie on the unit sphere (Sra, 2018). This distribution is described by a mean direction $\boldsymbol{\mu}$ with $\|\boldsymbol{\mu}\| = 1$, and a concentration parameter $\kappa \geq 0$. The vMF density function is defined as,

$$\text{vMF}(\boldsymbol{q}|\boldsymbol{\mu}, \kappa) = C_D(\kappa) \exp\left(\kappa \boldsymbol{\mu}^\mathsf{T} \boldsymbol{q}\right), \qquad \|\boldsymbol{\mu}\| = 1, \tag{3.3}$$

where C_D is the normalization constant

$$C_D(\kappa) = \frac{\kappa^{\frac{D}{2}-1}}{(2\pi)^{\frac{D}{2}} I_{\frac{D}{2}-1}(\kappa)}, \tag{3.4}$$

with $I_{\frac{D}{2}-1}(\kappa)$ being the modified Bessel function of the first kind. Like the Gaussian, from which the distribution was constructed, the von Mises-Fisher distribution is unimodal. To build a distribution that is antipodal symmetric, i.e. $p_{\psi}(\boldsymbol{q}|\boldsymbol{z}) = p_{\psi}(-\boldsymbol{q}|\boldsymbol{z})$, we define a mixture of antipodal vMF distributions (Hauberg et al., 2016),

$$p_{\psi}(\boldsymbol{q}|\boldsymbol{z}) = \frac{1}{2}\text{vMF}(\boldsymbol{q}|\boldsymbol{\mu}_{\psi}(\boldsymbol{z}), \kappa_{\psi}(\boldsymbol{z})) + \frac{1}{2}\text{vMF}(\boldsymbol{q}| - \boldsymbol{\mu}_{\psi}(\boldsymbol{z}), \kappa_{\psi}(\boldsymbol{z})), \tag{3.5}$$

where $\boldsymbol{\mu}$ and κ are parametrized as neural networks. This mixture model is conceptually similar to a Bingham distribution (Sra, 2018), but is easier to implement numerically.

3.3 Variational Inference

To train the VAE, we maximize an adapted evidence lower bound (ELBO) (2.8), defined as

$$\mathcal{L}_{ELBO} = \beta_1 \mathcal{L}_{\boldsymbol{x}} + \beta_2 \mathcal{L}_{\boldsymbol{q}} - \text{KL}\left(q_{\xi}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})\right), \tag{3.6}$$

$$\mathcal{L}_{\boldsymbol{x}} = \mathbb{E}_{q_{\xi}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\phi}(\boldsymbol{x}|\boldsymbol{z})\right], \tag{3.7}$$

$$\mathcal{L}_{\boldsymbol{q}} = \mathbb{E}_{q_{\xi}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\psi}(\boldsymbol{q}|\boldsymbol{z})\right], \tag{3.8}$$

where $\boldsymbol{x} \in \mathbb{R}^3$ and $\boldsymbol{q} \in \mathcal{S}^3$ represent the position and orientation of the end-effector, respectively. The scaling factors $\beta_1 > 0$ and $\beta_2 > 0$ balance the log-likelihood of position and orientation components. This approach is inspired by the β-VAE method (Higgins et al., 2017), which, despite not yielding a valid lower bound on the ELBO objective function, has been shown to produce high-quality results comparable to methods that provide a valid lower bound, thus it is commonly used in practice. Due to the quaternions antipodality, raw demonstration data may contain negative or positive values for the same orientation. So, we avoid any pre-processing step of the data by considering two vMF distributions that encode the same orientation at both sides of the hypersphere. Practically, we double the training data, by including $\boldsymbol{q}_n$ and $-\boldsymbol{q}_n$ for all observations $\boldsymbol{q}_n$. Note that as the Riemannian manifold is learned using task space data, the model is kinematics agnostic, which means that the generated motion may be used across different robots as long as the trajectory is reachable.

3.4 Induced Riemannian Metric

Our generative process is parametrized by a set of neural networks. Specifically, μ_{ϕ} and σ_{ϕ} are position mean and variance neural networks parameterized by $\boldsymbol{\phi}$, while μ_{ψ} and κ_{ψ} are neural networks parameterized by $\boldsymbol{\psi}$ that represent the mean and concentration of the quaternion distribution. It is important to note that we have used the same notation for the parameter values of the mean and variance networks. However, in practice, the parameters for these networks are not the same. Following Sec. 2.3, the Jacobians of these functions govern the induced Riemannian metric as,

$$\boldsymbol{M}(\boldsymbol{z}) = \boldsymbol{M}^{\boldsymbol{x}}_{\mu}(\boldsymbol{z}) + \boldsymbol{M}^{\boldsymbol{x}}_{\sigma}(\boldsymbol{z}) + \boldsymbol{M}^{\boldsymbol{q}}_{\mu}(\boldsymbol{z}) + \boldsymbol{M}^{\boldsymbol{q}}_{\kappa}(\boldsymbol{z}), \tag{3.9}$$

with

$$\boldsymbol{M}^{\boldsymbol{x}}_{\mu}(\boldsymbol{z}) = \boldsymbol{J}_{\mu_{\phi}}(\boldsymbol{z})^{\top}\boldsymbol{J}_{\mu_{\phi}}(\boldsymbol{z}), \tag{3.10}$$

$$\boldsymbol{M}^{\boldsymbol{x}}_{\sigma}(\boldsymbol{z}) = \boldsymbol{J}_{\sigma_{\phi}}(\boldsymbol{z})^{\top}\boldsymbol{J}_{\sigma_{\phi}}(\boldsymbol{z}), \tag{3.11}$$

$$\boldsymbol{M}^{\boldsymbol{q}}_{\mu}(\boldsymbol{z}) = \boldsymbol{J}_{\mu_{\psi}}(\boldsymbol{z})^{\top}\boldsymbol{J}_{\mu_{\psi}}(\boldsymbol{z}), \tag{3.12}$$

$$\boldsymbol{M}^{\boldsymbol{q}}_{\kappa}(\boldsymbol{z}) = \boldsymbol{J}_{\kappa_{\psi}}(\boldsymbol{z})^{\top}\boldsymbol{J}_{\kappa_{\psi}}(\boldsymbol{z}), \tag{3.13}$$

where $\boldsymbol{J}_{\mu_{\phi}}$, $\boldsymbol{J}_{\sigma_{\phi}}$, $\boldsymbol{J}_{\mu_{\psi}}$, $\boldsymbol{J}_{\kappa_{\psi}}$ are the Jacobians of the functions representing the position mean and variance, as well as the quaternion mean and concentration, respectively.

In practice, we want this Riemannian metric $\boldsymbol{M}(\boldsymbol{z})$ to take large values in regions with little or no data, so that geodesics avoid passing through them. We achieve this by using radial basis function (RBF) networks as our variance representation, whose kernels reliably extrapolate over the whole space (Arvanitidis et al., 2018). Since one of the main differences between Gaussian and von Mises-Fisher distributions is the representation of the data dispersion, the RBF network should consider a reciprocal behavior when estimating variance for positions. In summary, the data uncertainty is encoded by the RBF networks representing $\sigma^{-1}_{\phi}(\boldsymbol{z})$ and $\kappa_{\psi}(\boldsymbol{z})$, which affect the Riemannian metric through their corresponding Jacobians as in (3.9).

3.5 Geodesic Motion Skills

As explained previously, geodesics follow the trend of the data, and they are here exploited to reconstruct motion skills that resemble human demonstrations. In this

section, we describe geodesic computation in both settings, namely, where the VAE is trained on task space trajectories. Moreover, we explain how new geodesic paths, that avoid obstacles on the fly, can be obtained by metric reshaping. In particular, we exploit ambient space metrics defined as a function of the obstacle's configuration to locally deform the original learned Riemannian metric.

Last but not least, our approach can encode multiple-solution skills, from which new hybrid trajectories (not previously shown to the robot) can be synthesized. To elaborate, a multiple-solution task requires the robot to combine multiple demonstrated solutions, i.e. trajectories, in order to complete it. By being able to combine multiple solutions to generate a mixed novel trajectory, the robot can find a novel solution to a task despite that not a single demonstration provides a complete solution on its own. This can be particularly useful in complex or dynamic environments where the robot needs to be able to adapt and respond to changing conditions. We elaborate on each of these features in the sequel.

3.5.1 Generating Motion

Robot motion generation techniques that leverage demonstrations aim to replicate the demonstrated movement patterns with a high degree of accuracy, reliability, and efficiency. This is achieved by exploiting the demonstrations as a source of information about the range of (manifold of) possible movements that the robot should perform. The robot uses this information to generate similar movements on its own.

One way to represent the observed movement patterns mathematically is through the use of Riemannian manifolds. Riemannian manifolds allow us to describe data sub-spaces (defined by the demonstrations) within a larger space and provide analytical tools for calculating distances and angles within these sub-spaces. In the context of robot motion generation, a Riemannian manifold can be used to characterize the region of the state space covered by the demonstrations.

In this thesis, the Riemannian metrics that describe the structure of these manifolds are highly influenced by the data uncertainty and they are used to find the path between two points on the manifold, the so-called geodesic. In the context of Riemannian manifolds, geodesics are the curves that minimize the distance between two points on the manifold. It is important to keep in mind that the distance measurement is done on the manifold, using the distance metric of the physical space where the robot moves. This means that geodesics on a Riemannian manifold represent the most efficient paths

through the space represented by the manifold. Given that Riemannian manifolds represent the sub-space within which the demonstrations reside, it follows that the geodesics are equivalent to the optimal motion trajectories.

Specifically, the Riemannian metrics (2.11) and (4.3) tell us that geodesics are penalized for crossing through regions where the VAE predictive uncertainty grows. This implies that if a set of demonstrations follows a circular motion pattern, geodesics starting from arbitrary points on the learned manifold will also generate a circular motion (see Fig. 1.2–*left*). This behavior is due to the way that the metric $\boldsymbol{M}$ is defined: Our Riemannian metric $\boldsymbol{M}$ is characterized by low values where data uncertainty is low (and vice-versa). Since the geodesics minimize the energy of the curve between two points on $\mathcal{M}$, which is a function of $\boldsymbol{M}$, they tend to stay on the learned manifold and avoid outside regions. This property follows the common characteristics of motion generation techniques to reproduce motion learned from demonstrations and, makes us suggest that geodesics form a natural motion generation mechanism. Note that when using a Euclidean metric (i.e., an identity matrix), geodesics correspond to straight lines. Such geodesics certainly neglect the data manifold geometry. Note that geodesics do not typically follow a closed-form equation on these learned manifolds, as they are governed by the nonlinear differential system described in (2.5), and numerical approximations are required.

This can be done by direct minimization of curve length (Shao et al., 2018, Kalatzis et al., 2020), A* search (Chen et al., 2019), integration of the associated ODE (Arvanitidis et al., 2019), or various heuristics (Chen et al., 2018a). In this thesis, we compute geodesics on $\mathcal{M}$ by approximating them by cubic splines $c \approx \omega_\lambda(\boldsymbol{z}_c)$, with $\boldsymbol{z}_c = \{\boldsymbol{z}_{c_0}, \dots, \boldsymbol{z}_{c_K}\}$, where $\boldsymbol{z}_{c_k} \in \mathcal{Z}$ is a vector defining a control point of the spline over the latent space $\mathcal{Z}$. Given K control points, $K-1$ cubic polynomials ω_{λ_i} with coefficients $\lambda_{i,0}$, $\lambda_{i,1}$, $\lambda_{i,2}$, $\lambda_{i,3}$ have to be estimated to minimize its Riemannian length,

$$\mathcal{L}_{\omega_\lambda(\boldsymbol{z}_c)} = \int_0^1 \sqrt{\langle \dot{\omega}_\lambda(\boldsymbol{z}_c), \boldsymbol{M}(\omega_\lambda(\boldsymbol{z}_c))\dot{\omega}_\lambda(\boldsymbol{z}_c)\rangle}\mathrm{d}t. \tag{3.14}$$

The resulting geodesic c computed in $\mathcal{Z}$ is used to generate the robot motion by decoding it through the VAE networks μ_ψ and μ_ψ or μ_α depending on the ambient space setting. The obtained trajectory is then executed on the robot arm to reproduce the required skill. In the task space setting, the decoded geodesics can be deployed on the robot using a Cartesian impedance controller or inverse kinematics. In the joint

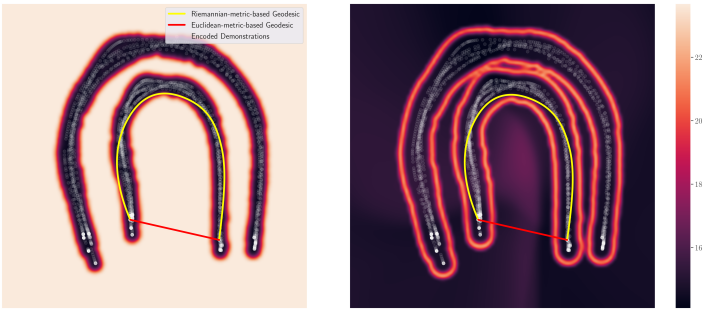


Figure 3.1: *Left*: The variance measure, *right*: The magnification factor of the Riemannian manifold learned from trajectories based on J and C English alphabet characters defined in $\mathbb{R}^2 \times S^2$. The semi-transparent white points depict the encoded training set, and the yellow curve depicts the geodesic in the latent space. The resulting manifold is composed of two similar clusters due to the antipodal encoding of the quaternions, where each cluster represents one side of the hyper-sphere. The yellow and red curves show the geodesics computed based on Riemannian and Euclidean metrics, respectively.

space setting, the decoded geodesics can be employed directly on the robot as a joint trajectory reference to be tracked by joint position or impedance controllers.

3.5.2 Geodesics in Task Space

In this section, we investigate the geodesic motion generation in task space. To illustrate the motion generation mechanism, we consider a simple experiment where the demonstration data at each time point is confined to $\mathbb{R}^2 \times S^2$, i.e. only two-dimensional positions and orientations are considered. We artificially create position data that follows a J shape and orientation data that follows a C shape projected on the sphere (see Fig. 3.2). We fit our VAE model to this dataset, and visualize the corresponding latent space in Fig. 3.1, where the top panel shows the latent mean embeddings of the training data with a background color corresponding to the predictive uncertainty. We see low uncertainty near the data, and high otherwise. The bottom panel of Fig. 3.1 shows the same embedding but with a background color proportional to $\log \sqrt{\det \boldsymbol{M}}$. This quantity, known as the magnification factor (Bishop et al., 1997), generally takes large values in regions where distances are large, implying that geodesics will try to avoid such regions. In Fig. 3.1, we notice that the magnification factor is generally low, except on the 'boundary' of the data manifold, i.e. in regions where the predictive variance grows. Consequently, we observe that Riemannian geodesics (yellow

curves in the figure) stay within the 'boundary' and hence resemble the training data patterns. In contrast, Euclidean geodesics (red curves in the figure) fail to stay in the data manifold. Our proposal is to use these geodesics on the learned manifolds as our robot motion generation mechanism.

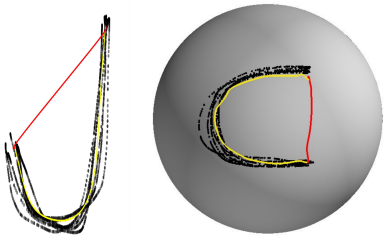


Figure 3.2: As an illustration, we consider synthetic data that belong to $\mathbb{R}^2 \times S^2$. The left panel depicts the J-shaped position data in $\mathbb{R}^2$ and the right panel shows the C-shaped orientation data on S^2. The yellow and red curves show the decoded geodesics depicted in Fig. 3.1, computed according to the Riemannian and Euclidean metrics, respectively.

Note that both panels in Fig. 3.1 depict two distinct horseshoe-like clusters, which is a result of the antipodality of the data in S^2. More precisely, the bimodal distribution of the antipodal quaternion data is encapsulated by these two clusters in the latent space. In practical settings, as long as the geodesic curve does not cross across clusters (both start and goal points belong to the same cluster), the quaternion sign is unchanged. We experimentally examine this in Section 3.6.

3.5.3 Obstacle Avoidance using Ambient Space Metrics

In the multiple-limb obstacle avoidance setting, rather than just using the end effector, the entire body of the robot is taken into account. This technique takes advantage of redundant solutions in joint space to find a robot configuration that avoids collisions. This is useful in situations where the end effector may be safe from an obstacle, but other parts of the robot's body (such as its links) are in danger of colliding with the obstacle. By providing multiple solutions in joint space during the demonstration phase, the robot can choose a configuration that avoids the obstacle and continue with its task. In cases where the demonstrations do not provide joint space solutions, a variety of obstacle avoidance techniques can be used to move the robot away from obstacles.

There are a variety of methods to incorporate obstacle avoidance into robotic systems. One commonly employed method is to utilize an off-the-shelf obstacle avoidance technique as post-processing of the generated trajectory. This approach involves first generating a trajectory, and then applying an obstacle avoidance technique to prevent collisions with obstacles present in the environment. This method can be relatively straightforward to implement and may effectively avoid obstacles. However, it can also lead to deviation from the intended motion and may not accurately reflect the demonstrations. It is worth noting that obstacle avoidance techniques that operate in joint space are often computationally intensive. Furthermore, this method cannot be applied in the latent space, as the obstacle avoidance technique must have knowledge of geometry and be able to generate trajectories on the manifold.

The technique that will be outlined in the following section offers a capability for efficient and effective obstacle avoidance for multiple limbs, without reliance on additional obstacle avoidance techniques.

Here we explain how we can reshape the learned metric to avoid obstacles in the task space setting, where only the robot end-effector is considered. Formally, we propose to define the ambient metric of the end-effector position to be

$$\boldsymbol{M}_p^{\boldsymbol{x}}(\boldsymbol{x}) = \left(1 + \zeta \exp\left(\frac{-\|\boldsymbol{x} - \boldsymbol{o}\|^2}{2r^2}\right)\right)\mathbb{I}_3, \quad \boldsymbol{x} \in \mathbb{R}^3, \tag{3.15}$$

where $\zeta > 0$ scales the cost, $\boldsymbol{o} \in \mathbb{R}^3$ and $r > 0$ represent the position and radius of the obstacle, respectively. The orientation metric is assumed to be the identity, denoted by $\boldsymbol{M}_{\mathcal{R}}$ (or $\boldsymbol{M}_{\mathcal{Q}}$). For example, if the orientation is represented by quaternions $\boldsymbol{q} \in \mathcal{S}^3 \subset \mathbb{R}^4$, then the corresponding metric on the tangent space $\mathcal{T}_{\boldsymbol{q}}\mathcal{S}^3$ (which is isomorphic to $\mathbb{R}^3$) is given by the 3×3 identity matrix $\mathbb{I}_3$. Therefore, the full metric is defined as follows,

$$\boldsymbol{M}_{\mathcal{X}} = \begin{pmatrix} \boldsymbol{M}_{\mathcal{P}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{M}_{\mathcal{R}} \text{ or } \boldsymbol{M}_{\mathcal{Q}} \end{pmatrix} \tag{3.16}$$

Under this new ambient metric, geodesics will generally avoid the obstacle, though we emphasize this is only a *soft* constraint. This approach is similar in spirit to CHOMP (Ratliff et al., 2009) except our formulation works along a low-dimensional learned manifold, whose solution is then decoded to the task space of the robot.

Under this ambient metric, the associated (reshaped) Riemannian metric of the latent space $\mathcal{Z}$ becomes,

$$\boldsymbol{M}(z) = \boldsymbol{M}^{x}_{\mu}(z) + \boldsymbol{M}^{x}_{\sigma}(z) + \boldsymbol{M}^{q}_{\mu}(z) + \boldsymbol{M}^{q}_{\kappa}(z), \tag{3.17}$$

$$\begin{aligned}
\text{with} \quad \boldsymbol{M}^{x}_{\mu}(z) &= \boldsymbol{J}_{\mu_\phi}(z)^{\top} \boldsymbol{M}^{x}_{\mathcal{X}}(\mu_\phi(z)) \boldsymbol{J}_{\mu_\phi}(z), \\
\boldsymbol{M}^{x}_{\sigma}(z) &= \boldsymbol{J}_{\sigma_\phi}(z)^{\top} \boldsymbol{M}^{x}_{\mathcal{X}}(\mu_\phi(z)) \boldsymbol{J}_{\sigma_\phi}(z), \\
\boldsymbol{M}^{q}_{\mu}(z) &= \boldsymbol{J}_{\mu_\psi}(z)^{\top} \boldsymbol{M}^{q}_{\mathcal{X}}(\mu_\psi(z)) \boldsymbol{J}_{\mu_\psi}(z), \\
\boldsymbol{M}^{q}_{\kappa}(z) &= \boldsymbol{J}_{\kappa\psi}(z)^{\top} \boldsymbol{M}^{q}_{\mathcal{X}}(\mu_\psi(z)) \boldsymbol{J}_{\kappa\psi}(z),
\end{aligned}$$

where $\boldsymbol{M}^{x}_{\mathcal{X}}$ and $\boldsymbol{M}^{q}_{\mathcal{X}}$ represent the position and orientation components of the obstacle-avoidance metric $\boldsymbol{M}_{\mathcal{X}}$, respectively. We emphasize that as the object changes position, the VAE does not need to be re-trained as the change is only in the ambient metric. As stated before, obstacles can be avoided only by the end-effector under this task space setting.

In the multiple-limb obstacle avoidance setting, rather than just using the end effector, the entire body of the robot is taken into account. This technique takes advantage of redundant solutions in joint space to find a robot configuration that avoids collisions. This is useful in situations where the end effector may be safe from an obstacle, but other parts of the robot's body (such as its links) are in danger of colliding with the obstacle. By providing multiple solutions in joint space during the demonstration phase, the robot can choose a configuration that avoids the obstacle and continue with its task. In cases where the demonstrations do not provide joint space solutions, a variety of obstacle avoidance techniques can be used to move the robot away from obstacles.

The technique that will be outlined in Sec. 4.3.1 offers a capability for efficient and effective obstacle avoidance for multiple limbs, without reliance on additional obstacle avoidance techniques.

3.5.4 Generating Geodesics on Discrete Manifolds

To generate robot motion, our approach requires computing geodesics, which can be done in several ways (Peyré et al., 2010). While this can be computed by solving an ordinary differential equation, this approach may not be suitable for real-time robotic applications due to its computational cost, as highlighted in previous studies (Arvanitidis et al., 2019). A commonly used alternative is to employ gradient descent to

minimize the curve length (Noakes et al., 2022). While this approach is straightforward and effective in some cases, it may not be optimal when operating in real-time. In our experiments, we found that the gradient descent approach often gets trapped in local minima, depending on a non-trivial initialization. As a consequence, multiple restarts may be required, which increases computational cost. We, therefore, consider an alternative discrete approach.

As we work with low-dimensional latent spaces, we here propose to simply discretizing them on a regular grid and use a graph-based algorithm for computing shortest paths. If this grid is sufficiently dense, this approach is globally optimal and computationally inexpensive. However, this approach is only feasible in low-dimensional latent spaces due to the curse of dimensionality. In particular, the memory requirements of the graph grow exponentially with dimension, and the short path search grows with the graph size. Our experiments are, however, concerned with low-dimensional latent spaces, where these limitations are negligible.

In detail, we create a uniform grid over the latent space $\mathcal{Z}$, and assign a weight to each edge in the graph corresponding to the Riemannian distance between neighboring nodes (see Fig. 3.4). Geodesics are then found using Dijkstra's algorithm (Cormen et al., 2009). This algorithm selects a set of graph nodes,

$$G_c = \left\{ \boldsymbol{g}_0, \boldsymbol{g}_1, \dots, \boldsymbol{g}_{S-1}, \boldsymbol{g}_S \right\}, \quad \boldsymbol{g}_s \in \mathbb{R}^D,$$

where $\boldsymbol{g}_0$ and $\boldsymbol{g}_S$ represent the start and the target of the geodesic in the graph, respectively. To select these points, the shortest path on the graph is calculated by minimizing the accumulated weight (cost) of each edge connecting two nodes, computed as in (2.3). To ensure a smooth trajectory, we fit a cubic spline ω_λ to the resulting set $\boldsymbol{G}_c$ by minimizing the mean square error. The spline computed in $\mathcal{Z}$ is finally used to generate the robot moves through the mean decoder networks: $\mu_{\boldsymbol{\theta}}$ and $\mu_{\boldsymbol{\psi}}$. The resulting trajectory can be executed on the robot arm to reproduce the required skill.

In order to verify the validity of the discrete geodesic approximation (graph-based) in comparison to the continuous approximation (gradient-based), a toy example scenario was devised for evaluating their respective accuracies. Three different experimental setups were compared, one using gradient descent and two using a graph-based approach. In the graph-based setting, the manifold was discretized, and the Dijkstra algorithm was used to find the shortest path. The discrete setups were also evaluated using two different resolutions to assess the effect of the grid density on the accuracy

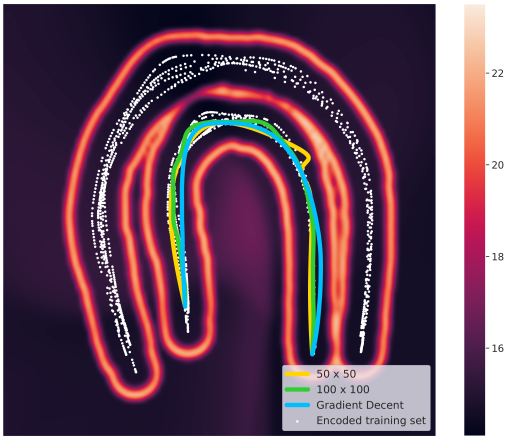


Figure 3.3: Comparison of geodesics generated under continuous and discrete setups. The results demonstrate that the geodesic computed using the 50×50 grid deviates unnecessarily from the data, likely due to the grid's relatively low resolution. In contrast, the geodesic generated using the 100×100 grid achieves accuracy comparable to that of the (continuous) gradient descent method.

and efficiency. We employed the same pre-trained model from the toy example presented in Section 3.5.2. Figure 3.3 displays the resulting geodesics. As observed, all methods were successful in approximating a geodesic that follows the trend of the data in the latent space. However, the geodesic generated using the 50×50 grid, depicted in yellow, exhibits an unnecessary deviation from the data, likely due to the relatively low resolution of the grid. Moreover, the geodesics generated using the 100×100 grid and gradient descent method exhibit a comparable level of accuracy.

Moreover, one issue with the graph-based approach is that dynamic obstacles imply that geodesic distances between nodes may change dynamically. To avoid recomputing all edge weights every time an obstacle moves we do as follows. Since the learned manifold does not change, we can keep a decoded version of the latent graph in memory (Fig. 3.4). This way we avoid querying the decoders at run-time. We can then find points on the decoded graph that are near obstacles and rescale their weights to take the ambient metric into account. Once the obstacle moves, we can reset the weights of points that are now sufficiently far from the obstacle.

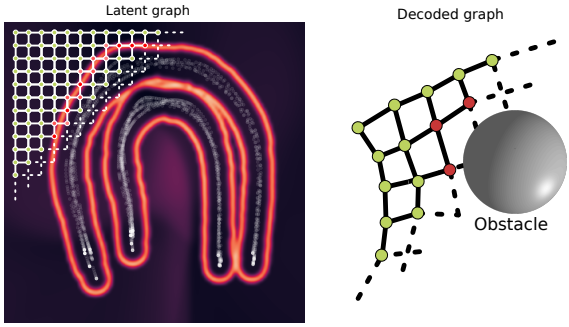


Figure 3.4: Concept drawing. *Left:* The latent space is discretized to form a grid graph consisting of linearly spaced nodes, with edge weights matching Riemannian distances. *Right:* To efficiently handle obstacles, the graph is decoded, such that obstacles can easily be mapped to latent space.

3.5.5 Architecture

In this section, we describe the VAE architecture under $\mathbb{R}^3 \times \mathcal{S}^3$ setting. The VAE architectures are implemented using PyTorch (Paszke et al., 2019a). This particular VAE network is designed to reconstruct end-effector poses in $\mathbb{R}^3 \times \mathcal{S}^3$. The overall architecture is depicted in Figure 3.5 with the different components that are required to correctly reproduce end-effector poses. In this architecture, both the decoder and encoder networks have two hidden layers with 200 and 100 neuron units (depicted as beige boxes) which output the mean and variance vectors (depicted as orange boxes). Furthermore, as previously explained, the variance and concentration parameters for position and quaternion data are estimated using RBF decoder networks with 500 kernels calculated by k-means over the training dataset (Arvanitidis et al., 2018) with predefined bandwidth. In this particular setup, the VAE uses a 2-dimensional latent space (depicted by the green box) to encode the 7-dimensional input vector (depicted as blue boxes on the left). Moreover, the VAE reconstructs the encoded input back to the ambient space (depicted as blue boxes on the right) using the decoder networks.

In this setting, the VAE is implemented using a single neural network as the decoder mean for the position and orientation, while the variance and concentration networks are implemented separately. As a result, the learned metric is defined as,

$$\boldsymbol{M}(\boldsymbol{z}) = \boldsymbol{M}_{\mu}^{\boldsymbol{x},\boldsymbol{q}}(\boldsymbol{z}) + \boldsymbol{M}_{\sigma}^{\boldsymbol{x}}(\boldsymbol{z}) + \boldsymbol{M}_{\kappa}^{\boldsymbol{q}}(\boldsymbol{z}), \tag{3.18}$$

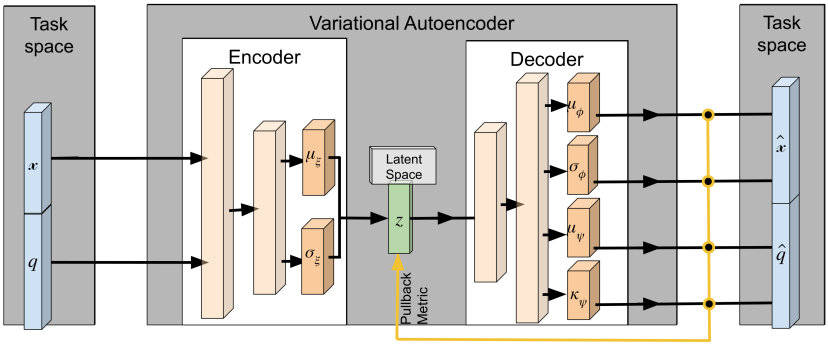


Figure 3.5: The VAE architecture under the task space setting. The blue, orange, green, and gray blocks correspond to ambient spaces, functions with trainable parameters, latent variables, and functions with fixed parameters. The arrows indicate the direction in which the data flows during the query.

$$
\begin{aligned}
\text{with} \quad \boldsymbol{M}_{\mu}^{x,q}(\boldsymbol{z}) &= \boldsymbol{J}_{\mu_{\boldsymbol{\phi}}}(\boldsymbol{z})^{\top} \boldsymbol{M}_{\mathcal{X}\mathcal{Q}}(\boldsymbol{z}) \boldsymbol{J}_{\mu_{\boldsymbol{\phi}}}(\boldsymbol{z}), \\
\boldsymbol{M}_{\mathcal{X}\mathcal{Q}}(\boldsymbol{z}) &= \begin{bmatrix} \boldsymbol{M}_{\mathcal{X}}(\mu_{\boldsymbol{\phi}}(\boldsymbol{z})) & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{M}_{\mathcal{Q}}(\mu_{\boldsymbol{\psi}}(\boldsymbol{z})) \end{bmatrix}, \\
\boldsymbol{M}_{\sigma}^{x}(\boldsymbol{z}) &= \boldsymbol{J}_{\sigma_{\boldsymbol{\phi}}}(\boldsymbol{z})^{\top} \boldsymbol{M}_{\mathcal{X}}(\mu_{\boldsymbol{\phi}}^{x}(\boldsymbol{z})) \boldsymbol{J}_{\sigma_{\boldsymbol{\phi}}}(\boldsymbol{z}), \\
\boldsymbol{M}_{\kappa}^{q}(\boldsymbol{z}) &= \boldsymbol{J}_{\kappa_{\boldsymbol{\psi}}}(\boldsymbol{z})^{\top} \boldsymbol{M}_{\mathcal{Q}}(\mu_{\boldsymbol{\psi}}(\boldsymbol{z})) \boldsymbol{J}_{\kappa_{\boldsymbol{\psi}}}(\boldsymbol{z}).
\end{aligned}
$$

where $\boldsymbol{J}_{\mu_{\boldsymbol{\phi}}} \in \mathbb{R}^{(D_{\mathcal{X}}+D_{\mathcal{Q}})\times d}$ is the Jacobian of the joint decoder mean network (position and quaternion), and $\boldsymbol{J}_{\sigma_{\boldsymbol{\phi}}} \in \mathbb{R}^{D_{\mathcal{X}}\times d}$ and $\boldsymbol{J}_{\kappa_{\boldsymbol{\psi}}} \in \mathbb{R}^{D_{\mathcal{Q}}\times d}$ are the Jacobians of the decoder variance and concentration RBF networks, respectively. Since the position and quaternion share the same decoder mean network, the output vector is split into two parts, accordingly. The quaternion part of the decoder mean is projected to the $\mathcal{S}^3$ to then define the corresponding von Mises-Fisher distribution as in (3.3). The yellow arrow in Fig. 3.5 shows the flow of the data from ambient space back to the latent space in order to compute the pullback metric.

The ELBO parameters β_1 and β_2 in (3.6) are found experimentally to guarantee good reconstruction of both position and quaternion data. It is worth pointing out that we manually provided antipodal quaternions during training, which leads to better latent space structures and reconstruction accuracy. The same architecture is used for all the experiments.

3.6 Experiments

The first set of experiments focuses on tasks where only the robot end-effector motion is relevant for the task, therefore the demonstrations are recorded in $\mathbb{R}^3 \times \mathcal{S}^3$.

3.6.1 Reach-to-grasp

The first set of experiments is based on a dataset collected while an operator performs kinesthetic demonstrations of a grasping skill. The grasping motion includes a 90° rotation when approaching the object for performing a side grasp (see Fig. 3.7). The demonstration trajectories start from the same end-effector pose, and they reach the same target position in the task space. To reproduce this grasping skill, we computed a geodesic in $\mathcal{Z}$ which is decoded to generate a continuous trajectory in $\mathcal{X}$, which closely reproduces the rotation pattern observed during demonstrations. Figure 3.7–*left* depicts the magnification factor related to the learned manifold. The semi-transparent white points correspond to the latent representation of the training set, and the yellow and blue curves are geodesics between points assigned from the start and endpoints of the demonstrations. The left panel in Fig. 3.7 shows geodesics in two different scenarios: The yellow geodesics in the right cluster start from the same pose and end up at different targets, while the blue geodesics in the left cluster start and end in different random targets. The results show that the method can successfully generate geodesics that respect the geometry of the manifold learned from the demonstration.

As expected, the magnification factor (Fig. 3.7–*left*) shows that the learned manifold is composed of two similar clusters, similar to the illustrative example in Fig. 3.1. We observed that this behavior emerged due to the antipodal encoding of the quaternions, where each cluster represents one side of the hyper-sphere. To confirm this, we generated a new geodesic, depicted in green in the right panel of Fig. 3.7, which is designed to cross the boundaries of the cluster (start and end locations belong to different clusters).

Figure 3.6 depicts the evolution of the quaternion components corresponding to the decoded green geodesic. As this geodesic curve crosses the clusters, the sign of the end-effector quaternion flips (highlighted by the black rectangle). It is worth emphasizing that by staying on the manifold and avoiding these boundaries, no post-processing of raw quaternion data is required during training or reconstruction. This can be easily guaranteed by monitoring the energy along the curves, which indicates when geodesics approach a high-energy region. For instance, the average energy of the blue and yellow

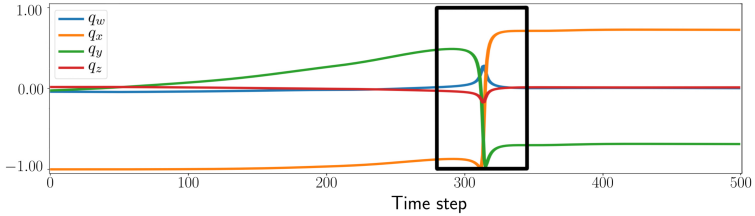


Figure 3.6: The evolution of the individual quaternion components of the green geodesic as it passes through high-energy regions.

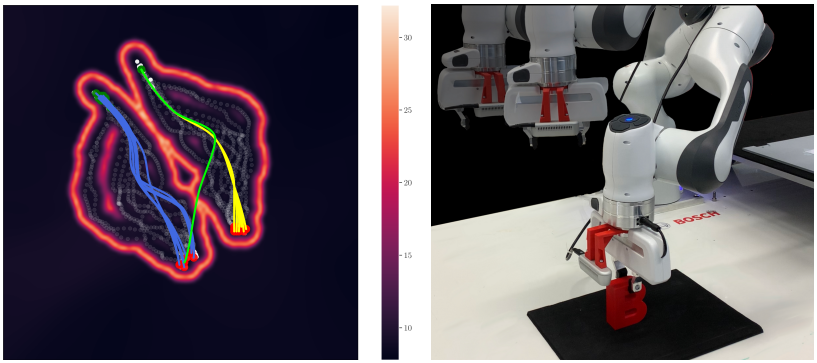


Figure 3.7: *Left*: The yellow curves in the right cluster show geodesics starting from the same points and ending up at random targets, and the blue curves in the left cluster connect random points on the manifold. The background depicts the magnification factor derived from the learned manifold, and the semi-transparent white points show the encoded training dataset. *Right*: The decoded geodesic employed on the robot.

geodesics are 7.50 and 10.51, respectively, while that of the green geodesic is 2.49×10^9. As a result, we can simply identify and avoid these geodesics.

Figure 3.7–*right* shows the reconstructed geodesic executed by the robot, where the overlapping images display the time evolution of the skill. It is easy to observe that the desired motion is successfully generated by our model. Note how the end-effector orientation evolves on the decoded geodesic in the ambient space, showing that the 90° rotation is properly encoded and reproduced using our approach.

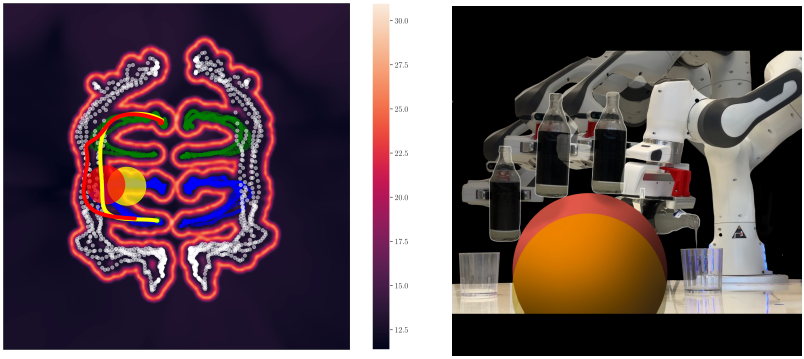


Figure 3.8: *Left*: Red and yellow curves depict geodesics at two different time frames performing dynamic obstacle avoidance with obstacles depicted as red and yellow circles. These geodesics also take advantage of a different group of demonstrations to generate a hybrid solution. The background depicts the magnification factor derived from the learned manifold with encoded demonstration sets depicted as dot clusters in white, green, and blue. *Right*: The decoded geodesics performed on the robot in different time steps.

3.6.2 Pouring

To evaluate our model on a more complicated scenario, we collected a dataset of pouring task demonstrations. The task involves grasping 3 bottles from 3 different positions and then pouring 3 cups placed at 3 different locations. The demonstrated trajectories cross each other, therefore providing a multiple-solution setting. As a result, with 3 sets of demonstrations, all 9 permutations for grasping any bottle from the table and then pouring at any cup are feasible, despite only a small subset of them being demonstrated.

The first feature we want to test in this setting is the obstacle avoidance capabilities via metric reshaping. To do so, we compute the ambient space metric in (3.15) based on a spherical obstacle that partially blocks the low-energy regions that the geodesics exploit to find a solution. This way the geodesics are forced to either use the low-energy regions that the individual demonstrations provide or improvise and find a hybrid novel path based on a subset of demonstrations. Figure 3.8–*left* shows the geodesic performing obstacle avoidance around a moving obstacle while following the geometry of the manifold. Two time instances of the obstacle are depicted as red and yellow circles in the latent space for illustration purposes.

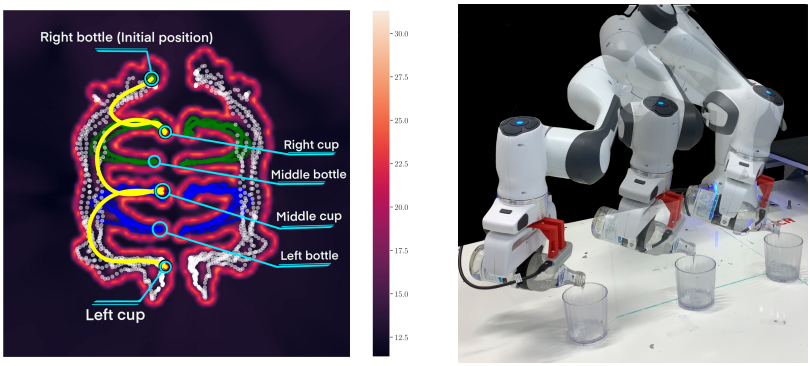


Figure 3.9: *Left*: The geodesic shown as the yellow curve combines the blue, green, and white demonstration groups to form a hybrid solution. This experiment uses three geodesics to pour all the cups using a single bottle. *Right*: The decoded geodesics performed on the robot are depicted by superimposing images from different time frames. The transparent robot arms depict the trace of the motion trajectory, which begins with pouring the right cup, the middle, and lastly the left cup.

The red and yellow curves represent geodesics avoiding the red and yellow obstacles, respectively. These curves correspond to one time frame of the adapted geodesic, showing how our method can deal with dynamic obstacles. Fig. 3.8–*right* shows the decoded geodesics performed on the robot, where transparent robot arms show the temporal evolution of the skill. In order to correctly perform obstacle avoidance, the parameter $\boldsymbol{x}$ in ambient space metric in (3.15) represents the position of the bottle when grasped and the obstacle radius r is modified to account for the bottle. To do so, we simplify the bottle geometry by approximating it using a sphere and add its radius r_{bot} to the radius of the obstacle as $r = r_{\text{obs}} + r_{\text{bot}}$, where r_{obs} is the obstacle radius. This prevents the bottle from colliding with the yellow and red spheres that represent the obstacle.

Figure 3.9–*left* shows the ability to leverage multiple-solution tasks to generate novel movements emerging as combinations of the observed demonstrations. Specifically, we generate a combination of three geodesics that can be used to pour all the cups with a single bottle. The geodesic (depicted as a yellow curve) starts from an initial point (depicted in green) in the white demonstrations group at the top, grasps the bottle, pours the first cup using the green demonstration group, then goes to the second cup using the blue-demonstrations group, and finally pours the third cup using the white-demonstration group again. Figure 3.9–*right* uses the same visualization technique to show the decoded geodesics employed on the robot by superimposing images from

different time frames. The task begins by pouring the right-side cup, then the center, and lastly the left-side cup. As observed, our approach generates the geodesics that properly switch among the demonstrated solutions to reproduce novel movements in ambient space and successfully pour all the cups in the given order.

Summary and Outlook

While generating geodesics in task space allows for intuitive reasoning and supports real-time obstacle avoidance at the end-effector level, it does not leverage the full kinematic redundancy of the robot. As a result, obstacle avoidance remains limited to the end-effector, which may be insufficient in cluttered or dynamic environments where the entire body of the robot is at risk of collision. To overcome this limitation, we extend our formulation to learn a Riemannian manifold in the joint space of the robot. This extension enables more flexible motion generation across the full kinematic chain and facilitates multiple-limb obstacle avoidance by accounting for the spatial configuration of all links. However, this formulation introduces additional challenges. The latent representation must capture the nonlinear structure of joint configurations, and learning a Riemannian manifold in joint space becomes more complex. Furthermore, integrating obstacle information requires processing through the forward kinematics of the robot, which significantly increases both computational and representational complexity.

In the next chapter, we address these challenges by formulating geodesic motion skills in joint space.

4 Geodesic Motion Skills in Joint Space

*This chapter is based on the paper **Reactive motion generation on learned Riemannian manifolds**, published in International Journal on Robotics Research (IJRR) in 2023 (Beik-Mohammadi et al., 2023)*

To extend the applicability of geodesic motion skills beyond task space, we now turn our attention to the robot's configuration space. The joint space $\mathbb{R}^\eta$, also known as *configuration space*, is another space to represent robot motion trajectories. [1]. In this space, each trajectory point is represented as the vector $\boldsymbol{\theta} = \left[\theta_1, \theta_2, \ldots, \theta_\eta\right] \in \mathbb{R}^\eta$, where η is the number of degrees of freedom of the robot. Learning motion skills in this space is known to be challenging as it is less intuitive to provide joint-level demonstrations. However, being able to learn and generate joint space movements is relevant as some tasks may demand specific robot postures. Moreover, joint space skills can be extended to provide whole-body obstacle avoidance. In this context, we formulate a Riemannian robot motion learning approach to generate collision-free joint space movements.

Previously, we computed a Riemannian metric in the latent space using the VAE decoder trained on task space demonstrations. Similarly, a new VAE architecture can be designed to compute a Riemannian metric from joint space demonstrations. We use this metric to compute geodesics and generate robot movements that resemble the demonstrations in joint space. This joint space approach also allows us to endow the robot with whole-body obstacle avoidance capabilities. By using ambient metrics, we can again reshape the learned metric to make the robot move away from obstacles in an online fashion. The ambient metrics exclusively use task space information of the obstacles and the robot body, in contrast to classical motion planning that often works in the configuration space. Note that the data manifold learned using joint space demonstrations is kinematics-dependent, meaning that the generated motion cannot be directly transferred to other robots with different kinematics.

[1] We did not model the robot joint space as a high-dimensional torus for simplicity. However, we showed that our approach can easily encode data on Riemannian manifolds as in the task space case.

4.1 Variational Inference

To train the joint space VAE, we maximize a modified version of the evidence lower bound (ELBO) (2.8), defined as

$$\mathcal{L}_{ELBO} = \mathcal{L}_{\theta} - \mathrm{KL}\left(q_{\xi}(\boldsymbol{z}_i|\boldsymbol{\theta}_i)||p(z)\right), \tag{4.1}$$

with,

$$\begin{aligned}\mathcal{L}_{\theta} &= \mathbb{E}_{q_{\xi}(\boldsymbol{z}_i|\boldsymbol{\theta}_i)}\left[p_{\mathcal{X}}(f_{\mathrm{FK}}(\boldsymbol{\theta})|\boldsymbol{z}_i)\right], \\ &= \mathbb{E}_{q_{\xi}(\boldsymbol{z}_i|\boldsymbol{\theta}_i)}\left[\log(p_{\Theta}(\boldsymbol{\theta}|\boldsymbol{z}_i)) - \log(\mathcal{V})\right],\end{aligned}$$

where $p_{\Theta}(\boldsymbol{\theta}|\boldsymbol{z}_i)$ and $p_{\mathcal{X}}(f_{\mathrm{FK}}(\boldsymbol{\theta})|\boldsymbol{z}_i)$ are the estimated conditional densities in the joint space Θ and task space $\mathcal{X}$, respectively. Also, $\mathcal{V}$ is the volume measure defined as

$$\mathcal{V} = \sqrt{\det(\boldsymbol{J}_{f_{\mathrm{FK}}}^{\mathsf{T}}(\mu_{\alpha}(\boldsymbol{z}_i))\boldsymbol{J}_{f_{\mathrm{FK}}}(\mu_{\alpha}(\boldsymbol{z}_i))}, \tag{4.2}$$

where $\boldsymbol{J}_{f_{\mathrm{FK}}}$ is the Jacobian of the forward kinematics f_{FK} given the joint configuration estimated by the VAE decoder μ_{α}. Furthermore, the generative distribution $p_{\Theta}(\boldsymbol{\theta}|\boldsymbol{z}_i) = \mathcal{N}(\mu_{\alpha}(\boldsymbol{z}_i), \mathbb{I}_{\eta}\sigma_{\alpha}(\boldsymbol{z}_i)^2)$ is parameterized by the VAE decoder mean $\mu_{\alpha}(\boldsymbol{z}_i)$ and variance $\sigma_{\alpha}(\boldsymbol{z}_i)$ networks. Note that the new ELBO formulation in (4.1) leverages the change of variable theorem (Deisenroth et al., 2020) to transform probability densities from joint to task space. As a result, the VAE is still trained using task space information, while the given demonstration trajectories are defined in joint space. This is motivated by the fact that most robot skills may still depend on task space variables (e.g. the manipulated objects pose), despite the same skill is also required to imitate particular robot postures.

As we are interested in whole-body obstacle avoidance, we can leverage the forward kinematics model to access the Cartesian position of different points on the robot (e.g., joint locations). Therefore, we use a set of $\boldsymbol{B}$ forward kinematic functions $f_{\mathrm{FK}}^{1:B}(\boldsymbol{\mu}_{\boldsymbol{\alpha}}^{1:\eta})$, where $\boldsymbol{\mu}_{\boldsymbol{\alpha}}^{1:n}$ is $n = 1, \ldots, \eta$ elements of the joint-values vector $\mu_{\alpha}(\boldsymbol{z})$, and $\boldsymbol{B}$ is the number of considered points on the robot. Note that for certain points on the robot structure, the forward kinematics only needs a subset of the joint values. For simplicity, we consider $\boldsymbol{B}$ to be equal to the number of robot joints plus the end-effector (i.e.

$B = \eta + 1$). Then, the full forward kinematic function f_{FK} is defined as,

$$\begin{aligned} f_{\text{FK}}(\boldsymbol{\mu}_{\boldsymbol{\alpha}}) &= \left[f^1_{\text{FK}}(\boldsymbol{\mu}^1_{\boldsymbol{\alpha}}), \dots, f^B_{\text{FK}}(\boldsymbol{\mu}^{1:\eta}_{\boldsymbol{\alpha}}), f^{ee}_{\text{FK}}(\boldsymbol{\mu}^{1:\eta}_{\boldsymbol{\alpha}})\right]^\mathsf{T}, \\ &= \left[\boldsymbol{p}_1, \dots, \boldsymbol{p}_B, \boldsymbol{p}_{\text{ee}}, \boldsymbol{q}_{\text{ee}}\right]^\mathsf{T}, \end{aligned}$$

where given the joint value vector $\boldsymbol{\mu}^{1:n}_{\boldsymbol{\alpha}}$ as input, all the functions compute the corresponding position $\boldsymbol{p}_m$ of the m-th point on the robot, except the last function f^{ee}_{FK} which also provides both the position $\boldsymbol{p}_{\text{ee}}$ and the orientation $\boldsymbol{q}_{\text{ee}}$ of the end-effector.

Furthermore, the volume measure $\mathcal{V}$ in (4.2) uses the Jacobian of the full forward kinematics function, which is defined as,

$$\boldsymbol{J}_{f_{\text{FK}}(\boldsymbol{\mu}_{\boldsymbol{\alpha}})} = \left[\boldsymbol{J}_{\boldsymbol{p}_1}, \dots, \boldsymbol{J}_{\boldsymbol{p}_B}, \boldsymbol{J}_{\boldsymbol{p}_{\text{ee}}}, \boldsymbol{J}_{\boldsymbol{q}_{\text{ee}}}\right]^\mathsf{T},$$

where $\boldsymbol{J}_{\boldsymbol{p}_i}$ and $\boldsymbol{J}_{\boldsymbol{q}_i}$ are the linear and angular components of the corresponding Jacobians.

4.2 Induced Riemannian Metric

With the new integrated forward kinematic layer, we can calculate a pullback metric that directly uses task space information. This adds an additional step in the formulation of the Riemannian metric, which now requires the Jacobian of the forward kinematics $\boldsymbol{J}_{f_{\text{FK}}}$ as well as the Jacobians of the VAE decoder $\boldsymbol{J}_{\mu_{\boldsymbol{\alpha}}}$ and $\boldsymbol{J}_{\sigma_{\boldsymbol{\alpha}}}$, computed from the mean and variance decoder networks. Using these two Jacobians the metric can be defined as,

$$\boldsymbol{M}^{\boldsymbol{\theta}}(\boldsymbol{z}) = \boldsymbol{M}^{\boldsymbol{\theta}}_{\mu_{\boldsymbol{\alpha}}}(\boldsymbol{z}) + \boldsymbol{M}^{\boldsymbol{\theta}}_{\sigma_{\boldsymbol{\alpha}}}(\boldsymbol{z}) \tag{4.3}$$

with,

$$\begin{aligned} \boldsymbol{M}^{\boldsymbol{\theta}}_{\mu_{\boldsymbol{\alpha}}}(\boldsymbol{z}) &= \left(\boldsymbol{J}_{f_{\text{FK}}}(\mu_{\boldsymbol{\alpha}}(\boldsymbol{z}))\boldsymbol{J}_{\mu_{\boldsymbol{\alpha}}}(\boldsymbol{z})\right)^\mathsf{T} \left(\boldsymbol{J}_{f_{\text{FK}}}(\mu_{\boldsymbol{\alpha}}(\boldsymbol{z}))\boldsymbol{J}_{\mu_{\boldsymbol{\alpha}}}(\boldsymbol{z})\right), \\ \boldsymbol{M}^{\boldsymbol{\theta}}_{\sigma_{\boldsymbol{\alpha}}}(\boldsymbol{z}) &= \left(\boldsymbol{J}_{f_{\text{FK}}}(\mu_{\boldsymbol{\alpha}}(\boldsymbol{z}))\boldsymbol{J}_{\sigma_{\boldsymbol{\alpha}}}(\boldsymbol{z})\right)^\mathsf{T} \left(\boldsymbol{J}_{f_{\text{FK}}}(\mu_{\boldsymbol{\alpha}}(\boldsymbol{z}))\boldsymbol{J}_{\sigma_{\boldsymbol{\alpha}}}(\boldsymbol{z})\right). \end{aligned}$$

Similarly to our Riemannian metric in task space, this new metric $\boldsymbol{M}^{\boldsymbol{\theta}}(\boldsymbol{z})$ takes large values in regions with little or no data so that geodesics avoid passing through them. Therefore, geodesic curves generated via (4.3) allow us to reproduce joint space robot skills.

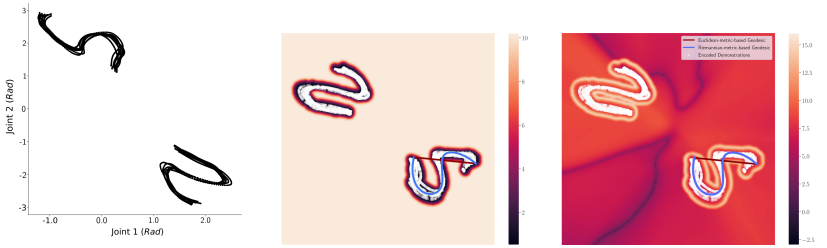


Figure 4.1: Illustration of joint space motion generation via geodesics. *Left*: S-shaped joint space demonstrations. *Middle*: the resulting variance measure. *Right*: The magnification factor of the learned Riemannian manifold. The semi-transparent white points depict the encoded training set and the blue curve represents the geodesic in the latent space. The resulting manifold is composed of two similar clusters due to the two different inverse-kinematics solutions for the task. The blue and red curves show the geodesics computed based on Riemannian and Euclidean metrics, respectively.

4.3 Geodesics in Joint Space

In this section, we investigate the geodesic computation for joint space movements. We use a toy example where a 2-DOF robot arm follows an S-shaped trajectory in task space using two different joint configurations (i.e., two different inverse-kinematics solutions), as shown in Fig. 4.1–left. We observe two sets of demonstrations in joint space that reproduce the same end-effector movements when applied to the robot. We can also see in the middle and right panels of Fig. 4.1 the geodesics computed using the Riemannian and Euclidean metrics, depicted as blue and red curves, respectively. The background of the middle panel illustrates the predictive uncertainty over the latent space $\mathcal{Z}$, where we again see low uncertainty near the data, and high otherwise. For completeness, the background in the right panel illustrates the magnification factor. The latent mean embedding of the training data is depicted as semi-transparent white points. Similar to the previous section, the geodesics generated using the Riemannian metric stay within the 'boundary' near the training data. Furthermore, it is easy to note that the learned manifold comprises two clusters, but unlike the previous task space example, these clusters arise from the two different joint space solutions provided in the training data. This indicates that the clusters in the learned manifold encapsulate the provided solutions in the demonstrations. When the number of clusters grows, the geodesic has a higher chance to travel among them to find a path with minimal energy as the high-energy regions may become narrow. However, unnecessary frequent switching among these clusters may often lead to jerky geodesics, therefore negatively

impacting the geodesics quality, particularly in robots with a high degree of freedom (e.g. DOF $\geq$ 7). Later in Sec. 4.5, we experimentally show that this issue can be alleviated by increasing the latent space dimensionality.

4.3.1 Obstacle Avoidance

Avoiding obstacles at the robot link level while performing motion skills requires considering the whole robot's kinematic structure. Classical motion planning methods model the geometry of the obstacles into the configuration space and later compute an obstacle-free path via sampling methods (Elbanhawi et al., 2014). In contrast, we take advantage of the forward kinematics layer (see Fig. 4.2), which provides us with task space poses of any point on the robot body, to compute an obstacle-avoidance ambient metric. Similar to the task space formulation presented previously, this ambient metric is then exploited to reshape the learned metric and generate modified geodesic curves that produce collision-free robot movements.

Specifically, we need to define a collection of points on the robot body $\boldsymbol{p}_1, \dots, \boldsymbol{p}_B$ with $\boldsymbol{p}_b \in \mathbb{R}^3$. These points are then used to compute the ambient space metric for obstacle-avoidance purposes. Therefore, a larger collection of points provides a more robust obstacle-avoidance performance at the cost of higher computational complexity. Given the set of points of interest, we compute an associated ambient metric following (3.15) with $\boldsymbol{x} = \boldsymbol{p}_b$. Similar to the task space setting, since the orientation of obstacles is not considered, the corresponding ambient space metric is an identity matrix. Finally, we form the whole ambient metric as $\boldsymbol{M}_{\mathcal{X}} = \text{blockdiag}(\left[\boldsymbol{M}_{\mathcal{X}}^{p_1}, \boldsymbol{M}_{\mathcal{X}}^{p_2}, \cdots, \boldsymbol{M}_{\mathcal{X}}^{p_B}\right])$, which is then used to reshape the learned metric of (4.3) as,

$$\boldsymbol{M}(\boldsymbol{z}) = \boldsymbol{M}_{\mu}^{\theta}(\boldsymbol{z}) + \boldsymbol{M}_{\sigma}^{\theta}(\boldsymbol{z}), \tag{4.4}$$

with,

$$\boldsymbol{M}_{\mu}^{\theta}(\boldsymbol{z}) = \left(\boldsymbol{J}_{f_{\text{FK}}}(\mu_{\alpha})\boldsymbol{J}_{\mu_{\alpha}}(\boldsymbol{z})\right)^{\mathsf{T}} \boldsymbol{M}_{\mathcal{X}} \left(\boldsymbol{J}_{f_{\text{FK}}}(\mu_{\alpha})\boldsymbol{J}_{\mu_{\alpha}}(\boldsymbol{z})\right),$$
$$\boldsymbol{M}_{\sigma}^{\theta}(\boldsymbol{z}) = \left(\boldsymbol{J}_{f_{\text{FK}}}(\mu_{\alpha})\boldsymbol{J}_{\sigma_{\alpha}}(\boldsymbol{z})\right)^{\mathsf{T}} \boldsymbol{M}_{\mathcal{X}} \left(\boldsymbol{J}_{f_{\text{FK}}}(\mu_{\alpha})\boldsymbol{J}_{\sigma_{\alpha}}(\boldsymbol{z})\right).$$

4.3.2 Architecture

Here, we describe our VAE network that reconstructs joint space movements in $\mathbb{R}^7$. The overall architecture is depicted in Figure 4.2 with different components. The input

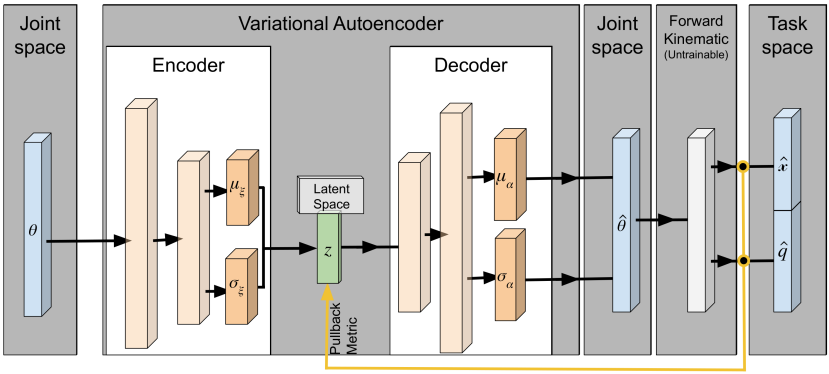


Figure 4.2: The VAE architecture under the joint space setting. The blue, orange, green, and gray blocks correspond to ambient spaces, functions with trainable parameters, latent variables, and functions with fixed parameters. The arrows indicate the direction in which the data flows during the query.

vector (depicted as the blue box on the left) is a joint-value vector representing a single configuration of the robot arm on a trajectory. This vector is fed to the encoder network with two hidden layers of 200 and 100 neuron units (depicted as beige boxes) which are the mean and variance vectors for the latent variable (depicted as orange boxes). Moreover, the variance RBF decoder network uses 500 kernels calculated by k-means over the training with predefined bandwidth. Under the joint space setting, the VAE uses a 3-dimensional latent space $\mathcal{Z}$ to encode the input vectors $\boldsymbol{\theta}$. As usual, the decoder network reconstructs the encoded inputs back to the joint space.

However, in order to access the task space information, necessary for whole-body obstacle avoidance, our architecture is integrated with a forward kinematics layer (depicted as the gray box). Note that this layer is predefined based on the robot arm kinematic model and does not change during training. We leverage this layer to compute task space information regarding multiple points on the robot (and not just the end-effector) given the input configuration vector $\boldsymbol{\theta}$. To implement this component we used the Python Kinematic and Dynamic Library PyKDL (Orocos, 2021).

It is worth noting that as this VAE architecture uses the forward kinematics layer during training, singular kinematic configurations need special attention. The main problem arises in the formulation of our ELBO in (4.1), which uses the volume measure computed as a function of the determinant of the Jacobian of the forward kinematics function. We can detect singularities when $\det(\boldsymbol{J}_{f_{\text{FK}}}(\mu_\alpha(\boldsymbol{z}_i))) = 0$, which may occur

due to random initialization of the VAE. To circumvent this issue and guarantee that the learning process is not disrupted, we simply add a small regularization term to the kinematic Jacobian.

Finally, we evaluate the obstacle avoidance capabilities in different scenarios where the obstacles partially or entirely obstruct the solutions in joint space. In these experiments, the ambient metric of (4.4) is formulated by considering all the joints on the robot, in addition to the end-effector. This ensures that the robot will avoid obstacles as long as they obstruct the solution for one or more joints or the end-effector. In other words, we do not consider points on the robot links lying between joints, since it was not necessary in our experiments. However, extra points on the robot links can be easily added to guarantee a more robust obstacle avoidance using the whole robot body.

4.4 Multiple-limb Obstacle Avoidance

Multiple-limb obstacle avoidance is a technique that aims to avoid obstacles by considering the entire robot body, rather than just the end-effector. This approach offers a more comprehensive way of obstacle avoidance as compared to techniques that focus exclusively on end-effector motion. To implement this method, we leverage redundant solutions in joint space from demonstrations. Therefore, by operating in the joint space, we can achieve redundancy at the joint level, which is essential for the success of multiple-limb obstacle avoidance. When working under the task space setting, it is not possible to achieve the redundancy in the joint level required for multiple-limb obstacle avoidance. Thus, attempts to avoid obstacles using multiple limbs may be unsuccessful. However, an off-the-shelf obstacle avoidance technique can be used to navigate the robot away from obstacles and complete the task under the task space setting.

To evaluate this, we used the trained model from Sec. 3.6.1 to show how likely an obstacle avoidance technique fails to provide an obstacle-free joint configuration. To do so, we decoded a 150×150 equidistant latent grid to access their corresponding input space states which represent the end-effector poses. Then, we used each pose to compute 100 different joint configurations using inverse kinematics. The obstacle was placed in a way that was close to the robot body but did not block any demonstration in the task space, as depicted in Fig. 4.3. Therefore, this setting requires multiple-limb obstacle avoidance in order to successfully perform the task. Each configuration was scored 1.0 even if a single point on the robot collided with the obstacle, otherwise, the score was 0.0.

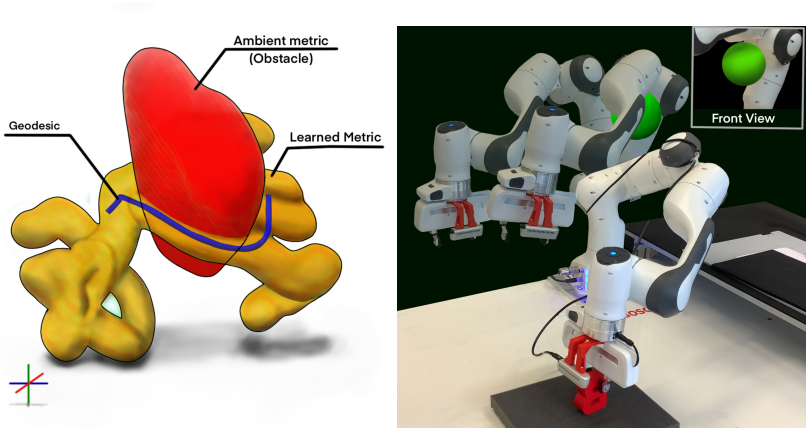


Figure 4.3: *Left*: The geodesic depicted as the blue curve avoids the obstacle by traveling through different demonstrations. *Right*: The decoded geodesic employed on the robot arm. Images from different time steps are superimposed to show the trace of the motion. The green sphere represents the obstacle in the ambient space.

In Fig. 4.4, the background shows the sum of these scores for each latent state over the latent grid, and the white points depict the encoded demonstrations in the latent space. The results show that in the regions close to demonstrations, there is up to 90 percent chance to generate a joint configuration that leads to a collision.

4.5 Experiments

In this section, we focus on tasks where joint-level motion patterns are relevant, and therefore the human teacher provides kinesthetic demonstrations in $\mathbb{R}^n$, where n is the number of DOF of the robot. When learning Riemannian metrics in this setting, we initially designed a 2-dimensional latent space for our VAE, which proved to be insufficient to encode the demonstrated motion patterns. Specifically, we analyze the capacity of the latent space to encode the skill manifold experimentally. To do so, we investigated the switching behavior of geodesics in $\mathcal{Z}$. In other words, overlap between low-energy regions in $\mathcal{Z}$, representing two or more different demonstration sets, may lead to unnecessary and frequent switches between these solutions when computing a geodesic. To put it differently, when computing a geodesic, switching between two demonstration sets is unavoidable when the total energy of the geodesic switching

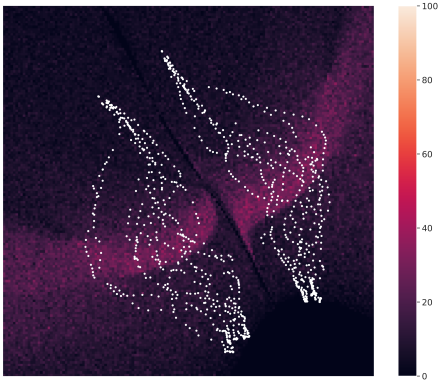


Figure 4.4: The background illustrates the frequency of failures by an obstacle avoidance technique to provide obstacle-free trajectories for the entire robot body. The white points depict the encoded demonstrations. The results indicate that the robot configurations generated using inverse kinematics in the regions close to the demonstrations are likely to result in collisions with some regions exceeding 90 collisions in 100 trials.

between them is less than the total energy without the switch. However, switching behaviors may be avoided by having high-energy regions among demonstration clusters in $\mathcal{Z}$. Furthermore, specifically under the joint space setting, the frequent switching in geodesics may lead to jittery motion in task space when applied to the robot. To illustrate this issue, we designed a simple experiment in which a robotic arm follows a circular pattern with its end-effector. The start and target configuration of the geodesic is selected from the same demonstrated trajectory, and the objective is to evaluate if the geodesic stays on the low-energy regions corresponding to the same demonstrated trajectory.

Fig. 4.5–*left* shows a geodesic curve computed in the 2-dimensional latent space depicted as the yellow curve. This geodesic exhibits unnecessary switches among different solutions (i.e. circular white demonstrations). Therefore, when the decoded geodesic is deployed on the robot, it results in jerky movements and undesirable back-and-forth motions. To solve this issue, we evaluated the same experiment using a 3-dimensional latent space. Figure 4.5–*right* shows the magnification factor of the metric learned using the same training data but in a 3–dimensional latent space. This magnification factor shows several torus-like clusters, representing separate demonstration groups instead of collapsing them into a plane, as in the 2-dimensional case. Moreover, the

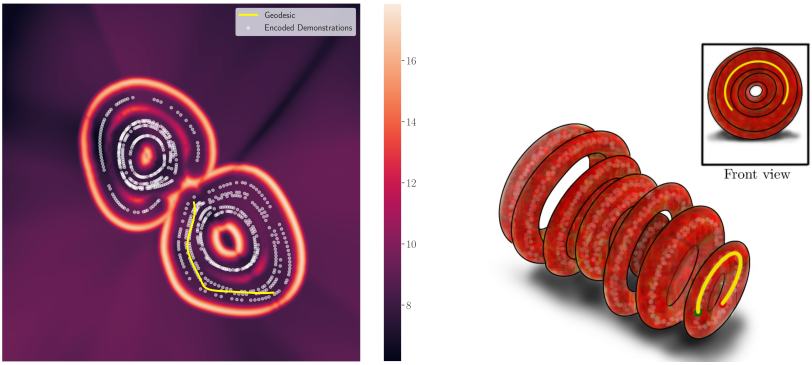


Figure 4.5: *Left*: The geodesics calculated in 2–dimensional latent space, depicted as the yellow curve, reveal several unnecessary transitions between different solutions when connecting two points in the same demonstration. *Right*: Geodesics computed in 3–dimensional latent space, shown by the yellow curve, shows that the geodesic does not switch between clusters.

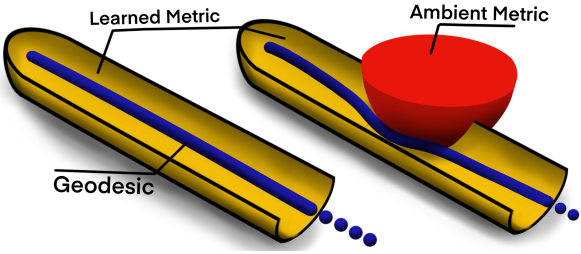


Figure 4.6: Concept drawing. *Left*: The hollow tube represents the metric, with low-energy regions inside surrounded by high-energy boundaries. The geodesic depicted as the blue curve travels through the low-energy regions. *Right*: The hollow tube is partially blocked by a solid high-energy region corresponding to the ambient space metric representing the obstacle. The geodesic depicted as the blue curve successfully travels through the low-energy regions meanwhile avoiding the obstacle.

resulting geodesic (depicted as the yellow curve) does not switch among solutions, which provides stable and smooth robot end-effector movements when decoded.

To provide further details on the learned manifold in the 3-dimensional latent space, we create an illustration shown in Fig. 4.6, where the learned metric is shown as yellow hollow tubes. Their inner part encodes low-energy regions which are surrounded by high-energy boundaries. Figure 4.6 displays a horizontal cut to show the inner part of the learned metric. To illustrate the obstacles, the right hollow tube is partially

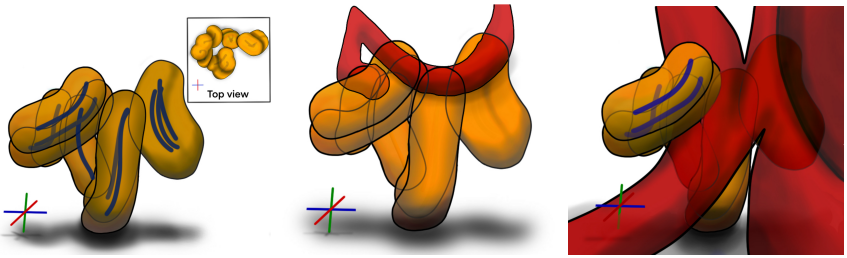


Figure 4.7: *left*: The magnification factor in 3–dimensional latent space contains hollow tubes representing the learned Riemannian metric. These hollow tubes contain low-energy regions surrounded by high-energy boundaries. Several geodesics are calculated and visualized in blue. The manifold's low-energy areas are rendered transparent. *middle*: The same magnification factor when an obstacle is introduced in such a manner that all viable solutions in the ambient space are blocked. Due to the fact that the obstacle introduces a high energy zone (in red) that goes through all of these hollow tubes, none of the geodesics are feasible, therefore no geodesic is shown in this plot. *right*: The same magnification factor when an obstacle is introduced in such a manner that partially obstructs the solutions in the ambient space. Several geodesics (blue curves) were left outside of the obstacle region on the left side of the panel.

blocked by the red sphere representing the ambient space metric in the latent space $\mathcal{Z}$, which is a solid high-energy region. Additionally, the figure depicts a geodesic curve traveling successfully along both tubes, showcasing a collision-free trajectory at the right-side plot.

4.5.1 Reach-to-grasp

Similar to the task space setting, we used the reach-to-grasp task to evaluate our approach under the joint space setting. In this case, the demonstrations are quite similar at the end-effector level but differ at the joint space, as we exploited the kinematic robot redundancy to provide different joint trajectories. Figure 4.7–*left* shows the magnification factor in a 3-dimensional latent space, where it can be seen that the learned metric corresponds to several connected and separated hollow tubes. As mentioned previously, these hollow tubes have low-energy inner regions surrounded by high-energy boundaries, analogous to the learned metric in a 2-dimensional latent space. As shown in the figure, the generated geodesics stay inside the tubes and avoid crossing the boundaries.

Figure 4.8–*left* shows the robot executing the decoded geodesic by applying the joint values directly on the robot using a joint position controller. We can observe that the

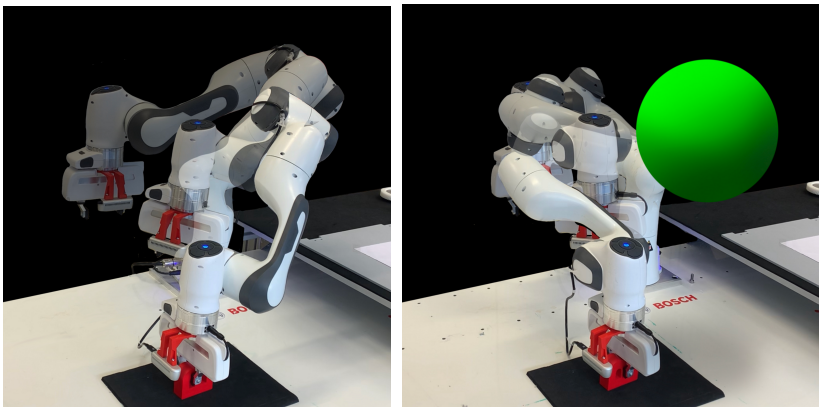


Figure 4.8: *Left*: The decoded geodesic is employed on the robot arm when no obstacle is present in the environment. *Right*: The decoded geodesic employed on the robot arm with an obstacle partially obstructing the solutions. The obstacle is depicted as the green sphere.

decoded geodesic is able to generate the demonstrated grasp motion with 90° rotation during the approaching part. Note that the start and end points of the geodesics are extracted from the demonstrations.

Figure 4.7–*middle* displays the magnification factor where an obstacle is placed in such a way that all possible solutions in joint space are blocked (e.g. obstacle placed on the common target of all the demonstrations). Note that the obstacle introduces a high energy zone (depicted in red) that passes through all of the hollow tubes representing the learned manifold, therefore, none of the geodesics are feasible. The feasibility of the geodesics is assessed by evaluating the total energy of the curve. If a geodesic passes through an obstacle, its energy will suddenly increase, providing an indication of a potential collision. Additionally, Fig. 4.7–*right* illustrates the same learned metric but reshaped using a different ambient space metric. We can now see that the obstacle partially obstructs the possible solutions, and therefore some few geodesics (depicted as blue curves) are still able to successfully generate obstacle-free movements. Figure 4.8–*right* shows the robot executing the decoded geodesic using a joint position controller while avoiding the obstacle.

We designed another experiment to showcase how the multiple-solution capabilities can be leveraged to generate collision-free movements. If the different demonstrated joint space trajectories sufficiently overlap, the learned manifold will be characterized

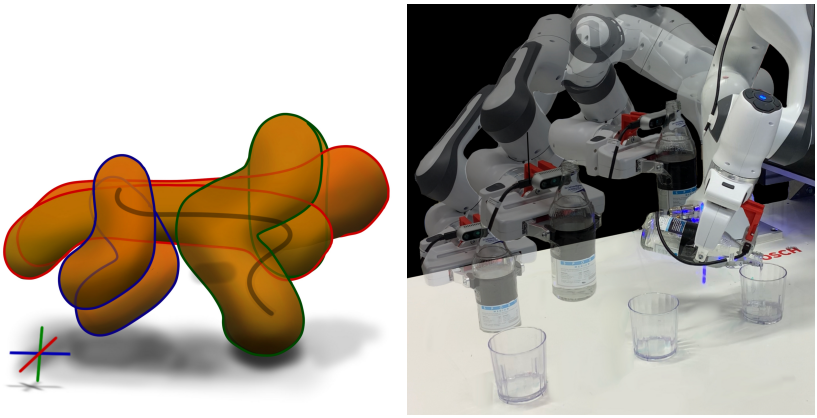


Figure 4.9: *Left*: The magnification factor in 3–dimensional latent space representing the learned Riemannian metric. This metric contains low-energy regions surrounded by high-energy boundaries depicted as red hollow tubes. The geodesic curve is depicted in black. The outline color of each tube (green, red, or blue) indicates which demonstration group it belongs to. *Right*: The decoded geodesic employed on the robot.

by several overlapping low-energy regions (i.e. hollow tubes), which geodesic curves can travel through. As a result, if the obstacles partially block the learned manifold, the geodesic may still travel among solutions to generate new movements out of combinations of the provided demonstrations. To show this behavior, we used a different demonstration dataset where we accounted for several overlapping solutions starting from the same joint configuration. Figure 4.3–*left* shows the magnification factor of the learned Riemannian metric in the 3–dimensional latent space. The learned manifold and the ambient metric can be distinguished visually based on their energy values. The ambient metric, which represents the obstacle regions (depicted in red), encodes high-energy zones. The learned manifold (depicted in yellow) is characterized by several entangled hollow tubes. The geodesic, shown as the blue curve, begins from the common start configuration on the left and successfully navigates around the obstacle to reach the target on the right side. Figure 4.3–*right* displays the decoded geodesic deployed directly on the robot arm using a joint position controller. Note how the robot arm avoids the obstacle by carefully maneuvering around it while successfully performing the grasping skill.

4.5.2 Pouring

To evaluate the method in a more complex scenario, we also performed the pouring task under the joint space setting. Similar to the task space experiment, the demonstrations also overlap in joint space. For this specific experiment, we mainly focus on evaluating the multiple-solution trajectories.

Figure 4.9–*left* shows the magnification factor in 3-dimensional latent space showing entangled hollow tubes, which represent the learned Riemannian metric. Each hollow tube encodes low-energy regions surrounded by high-energy boundaries, the latter outlined by green, red, and blue solid lines. Each color represents one group of demonstrations. Since the magnification factor learned with the original dataset of 5 demonstrations per group (corresponding to the bottle's initial position) is very hard to visualize, we opt for simplicity and used a subset of 2 demonstrations per bottle. As shown, the geodesic (depicted as the black curve) starts from a point in the second demonstration group (green border) and switches to the first group (red border) to reach the target located in the third group (blue boundary). Figure 4.9–*right* shows that the decoded geodesic employed on the robot successfully performs the pouring task. It is also evident the geodesic uses a multiple-solution strategy to reproduce a new trajectory that was not explicitly demonstrated to the robot in the training phase.

Summary and Outlook

This chapter introduced a Riemannian perspective for motion generation, where human demonstrations are modeled as geodesics on a learned manifold. This framework not only enables smooth and adaptable motion reproduction but also supports real-time obstacle avoidance through metric reshaping. Additionally, we demonstrated how this approach naturally handles multiple-solution tasks, allowing for the generation of novel trajectories not explicitly demonstrated during training. While this geometric strategy provides a powerful mechanism for capturing the spatial characteristics of motion, it does not encode temporal dynamics. Some tasks, however, require understanding how motion evolves over time, which demands a representation of dynamic features. These challenges motivate the transition to the next chapter, where we focus on learning stable dynamical systems capable of encoding both spatial and temporal aspects of robot motion.

5 Contractive Dynamics

*This chapter is based on the paper **Neural Contractive Dynamical Systems**, published in International Conference on Robot Learning (ICLR) in 2024 (Beik-Mohammadi et al., 2024), and the paper **Extended Neural Contractive Dynamical Systems: On Multiple Tasks and Riemannian Safety Regions**, published in International Journal on Robotics Research in 2024 (Beik-Mohammadi et al., 2025).*

Dynamical systems offer a powerful framework for modeling robot motion as time-evolving trajectories. However, ensuring stability and robust behavior – specially under perturbations and in dynamic environments – remains a fundamental challenge. Traditional learning-based dynamical systems often struggle to provide reliable guarantees on long-term behavior, particularly when extrapolating beyond demonstration data. To address this, we draw on principles from contraction theory, which provides strong stability guarantees by ensuring that all trajectories of the system converge toward one another over time. In this chapter, we introduce the Neural Contractive Dynamical System (NCDS) method, a novel approach for learning stable vector fields with built-in contraction properties. Our main goal is to design a flexible neural architecture that is guaranteed to always output a contractive vector field.

5.1 Neural Contractive Dynamical Systems

From Definition 2.1, we seek a flexible neural network architecture, such that the symmetric part of its Jacobian is negative definite. We consider the dynamical system $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$, where $\boldsymbol{x} \in \mathbb{R}^D$ denotes the system's state and $f : \mathbb{R}^D \to \mathbb{R}^D$ is a neural network. Note that it is not trivial to impose a negative definiteness constraint on a network's Jacobian without compromising its expressiveness. To overcome this challenge, we first design a neural network $\hat{\boldsymbol{J}}_f$ representing directly the Jacobian of our final network. This produces matrix-valued negative definite outputs. The final neural network, parametrizing $f_{\boldsymbol{\theta}}$, will then be formed by integrating the Jacobian network.

Specifically, we define the Jacobian as,

$$\hat{\boldsymbol{J}}_f(\boldsymbol{x}) = -(\boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{x})^\top \boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{x}) + \varepsilon \, \mathbb{I}_D), \tag{5.1}$$

where $\boldsymbol{J}_{\boldsymbol{\theta}} : \mathbb{R}^D \to \mathbb{R}^{D \times D}$ is a neural network parameterized by $\boldsymbol{\theta}$, $\varepsilon \in \mathbb{R}_+$ is a small positive constant, and $\mathbb{I}_D$ is an identity matrix of size D. Intuitively, $\boldsymbol{J}_{\boldsymbol{\theta}}$ can be interpreted as the (approximate) square root of $\hat{\boldsymbol{J}}_f$. Clearly, $\hat{\boldsymbol{J}}_f$ is negative definite as all eigenvalues are bounded from above by $-\varepsilon$.

Next, we take inspiration from Lorraine et al. (2019) and integrate $\hat{\boldsymbol{J}}_f$ to produce a function f, which is implicitly parametrized by $\boldsymbol{\theta}$, and has Jacobian $\hat{\boldsymbol{J}}_f$. The fundamental theorem of calculus for line integrals tells us that we can construct such a function by a line integral,

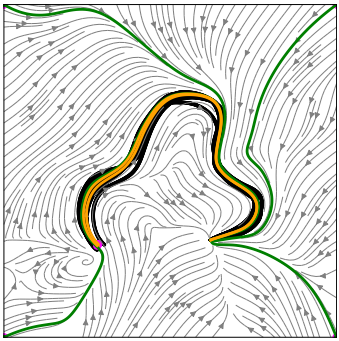


Figure 5.1: The learned vector field (grey) and demonstrations (black). Yellow and green trajectories show path integrals starting from demonstration starting points and random points, respectively.

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}) = \dot{\boldsymbol{x}}_0 + \int_0^1 \hat{\boldsymbol{J}}_f \left(c\left(\boldsymbol{x}, t, \boldsymbol{x}_0\right)\right) \dot{c}(\boldsymbol{x}, t, \boldsymbol{x}_0) \mathrm{d}t, \tag{5.2}$$

$$\begin{aligned} \text{with} \quad & c(\boldsymbol{x}, t, \boldsymbol{x}_0) = (1-t)\, \boldsymbol{x}_0 + t\boldsymbol{x}, \\ & \dot{c}(\boldsymbol{x}, t, \boldsymbol{x}_0) = \boldsymbol{x} - \boldsymbol{x}_0, \end{aligned}$$

where $\boldsymbol{x}_0$ and $\dot{\boldsymbol{x}}_0 = f(\boldsymbol{x}_0)$ represent the initial conditions of the state variable and its first-order time derivative, respectively. The input point $\boldsymbol{x}_0$ can be chosen arbitrarily

(e.g. as the mean of the training data or it can be learned), while the corresponding function value $\dot{\boldsymbol{x}}_0$ has to be estimated along with the parameters $\boldsymbol{\theta}$. Therefore, given a set of demonstrations denoted as $\mathcal{D} = \{\boldsymbol{x}_i, \dot{\boldsymbol{x}}_i\}$, our objective is to learn a set of parameters $\boldsymbol{\theta}$ along with the initial conditions $\boldsymbol{x}_0$ and $\dot{\boldsymbol{x}}_0$, such that the integration in (5.2) enables accurate reconstruction of the velocities $\dot{\boldsymbol{x}}_i$ given the state $\boldsymbol{x}_i$. This is achieved through the velocity reconstruction loss,

$$\mathcal{L}_{\text{vel}} = \frac{1}{N} \sum_{i=1}^{N} \left\| \dot{\boldsymbol{x}}_i - \hat{\dot{\boldsymbol{x}}}_i \right\|^2, \tag{5.3}$$

where $\dot{\boldsymbol{x}}_i$ denotes the demonstrated velocity, $\hat{\dot{\boldsymbol{x}}}_i$ is the predicted velocity, and N represents the number of data points. This process is shown in block B of the architecture in Fig. 6.1.

Note that the integral in (5.2) resembles the neural ordinary differential equations (Chen et al., 2018c), with the subtle difference that it is a second-order equation as the outcome pertains to the *velocity* at state $\boldsymbol{x}$. We can, thus, view this system as a second-order neural ordinary equation (Norcliffe et al., 2021) and solve it using off-the-shelf numerical integrators. The resulting function $f_{\boldsymbol{\theta}}$ will have a negative definite Jacobian for any choice of $\boldsymbol{\theta}$ and is consequently contractive by construction. In other words, we can control the extrapolation behavior of the neural network that parameterizes our dynamical system f via the negative-definiteness of $\hat{\boldsymbol{J}}_f(\boldsymbol{x})$.

We call this the *neural contractive dynamical system (NCDS)*. This approach offers two key benefits: *(1)* it allows the use of any smooth neural network $\boldsymbol{J}_{\boldsymbol{\theta}}$ as the base model, and *(2)* training can be performed with ordinary *unconstrained* optimization, unlike previous approaches. Figure 5.1 shows an example vector field learned by NCDS, which is clearly highly flexible while still providing global contractive stability guarantees.

With the introduction of NCDS, we propose two ways to improve it by focusing on the formulation of the Jacobian $\hat{\boldsymbol{J}}_f(\boldsymbol{x})$, as it is the key element that influences the behavior of the system. The formulation of the Jacobian in (5.1) is characterized by two main components: the symmetric matrix $\boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{x})^\top \boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{x})$, and the regularization term $\varepsilon \, \mathbb{I}_D$. Each of these components is crucial in determining the behavior of the model on the data support, and more importantly, its ability to generalize outside of it.

The NCDS formulation is designed to effectively learn complex contractive dynamical systems. In the following sections, we will investigate ways to refine it by applying different regularization techniques and exploring alternative Jacobian formulations.

5.2 Regularization

The primary objective of the regularization term ϵ is to ensure that the Jacobian matrix $\hat{\boldsymbol{J}}_f$ associated with NCDS is not semi-definite, thus breaking the contraction guarantees. Notice that the eigenvalues of the symmetric part of the Jacobian indicate how the system contracts or expands along different eigenvectors. Moreover, the

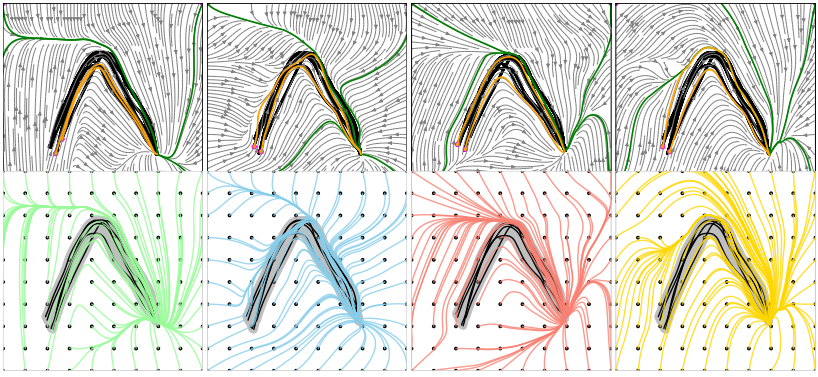


Figure 5.2: Comparison of 2D vector fields learned by using the following regularization approaches: **(a)** constant regularization, **(b)** state-independent regularization (using basic optimization), **(c)** state-dependent regularization (using a neural network), and **(d)** eigenvalue regularization. In the *top row*, the gray background illustrates the learned vector fields, with colored trajectories (integral curves) starting from the initial demonstration point and from the four corners of the figure. In the *second row*, 100 integral curves are shown, each initiated from an equidistant grid that *encompasses* the demonstration region. This region (shaded gray) is defined by the convex hull around the demonstrations (black curves).

contraction rate of a system is determined by its eigenvalue closest to zero (i.e., the largest eigenvalue as all eigenvalues are negative), representing the minimum rate of convergence that the system exhibits along any direction. The regularization method, here referred to as *constant regularization*, adds a small constant to the diagonal terms of the Jacobian $\boldsymbol{J}_\theta^\top \boldsymbol{J}_\theta$. Therefore, the final learned eigenvalues are influenced by both the neural network parameters and the applied regularization.

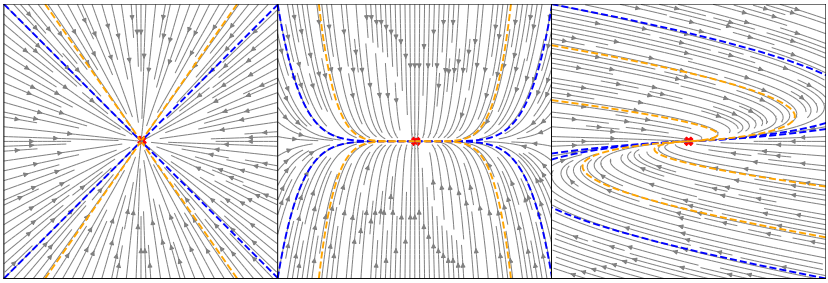


Figure 5.3: The effect of eigenvalue differences and asymmetry of the Jacobian on the contraction behavior of the vector field. *Left*: In a contractive dynamical system with a symmetric Jacobian, when the eigenvalues are equal, the system contracts uniformly in all directions. *Middle*: In a contractive dynamical system with a symmetric Jacobian, when the eigenvalues are different, the system contracts more rapidly in the direction associated with the larger eigenvalue. *Right*: An example of a contractive dynamical system with an asymmetric Jacobian.

Moreover, it indirectly determines the system's contraction rate by providing an upper bound on the eigenvalues. Note that when these eigenvalues are identical, the system exhibits isotropic contraction, which results in integral curves that are strictly straight as they converge directly to the fixed point, rather than following any curved trajectory (see Fig. 5.3–*left*). Even under the former condition, the system remains contractive. Interestingly, by increasing the disparity between the eigenvalues, the system converge more rapidly along certain axes (Fig. 5.3–*middle*).

Formally, let $\hat{\boldsymbol{J}}_f$ denote the Jacobian matrix with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_D$. We can obtain different contractive behaviors as a function of λ_i, as follows: (1) When $\lambda_1 = \lambda_2 = \ldots = \lambda_D$, the system contracts uniformly in all directions (see Fig. 5.3–*left*); (2) When $\lambda_i > \lambda_j$, the system exhibits faster contraction along the direction associated with the larger eigenvalue λ_i, and slower contraction in the direction corresponding to the smaller eigenvalue λ_j. As illustrated in Fig. 5.3–*middle*, this difference in contraction rates leads to the characteristic curved trajectories of the path integrals, as the system contracts more rapidly along the eigenvector of λ_i and more gradually along that of λ_j. We define the *contraction ratio* of the system as the absolute difference between these eigenvalues, at any point $\boldsymbol{x}_i$. Our goal is not to employ the contraction ratio as a fundamental concept in contraction systems, but rather as a bias in the learning process to promote stronger overall contraction. This contrasts with the conventional *contraction rate*, which only characterizes the slowest contracting eigenvector and fails to provide insight into the overall system's contractive behavior.

Table 5.1: The contraction rate and maximum contraction ratio computed over the equidistant grids.

Metrics	Contraction rate	Contraction ratio (max)
Constant reg.	−0.010	1.393
State-independent reg.	**−0.012**	**2.489**
State-dependent reg.	-10^{-4}	0.796
eigenvalue reg.	-10^{-4}	0.801

Given this analysis, we propose to better control the system's contraction using the contraction ratio and through regularization by considering the following three approaches: *(1) state-independent regularization vector, (2) state-dependent regularization vector*, and *(3) eigenvalue regularization.* We now proceed to explain these different regularization approaches for NCDS.

5.2.1 Regularization Vector

Here we treat $\varepsilon \in \mathbb{R}^D$ as a vector rather than a constant. Consequently, the Jacobian is reformulated as follows,

$$\hat{\boldsymbol{J}}_f(\boldsymbol{x}) = -(\boldsymbol{J}_\theta(\boldsymbol{x})^\top \boldsymbol{J}_\theta(\boldsymbol{x}) + \mathrm{diag}(\boldsymbol{\varepsilon})). \tag{5.4}$$

We propose two distinct methods to learn the vector $\boldsymbol{\varepsilon}$. For a more comprehensive comparison, we employed several metrics, the details of which are provided in Sec. 6.4.
State-independent: In this method, $\boldsymbol{\varepsilon}$ is assumed to be independent of the system's state $\boldsymbol{x}$. The vector $\boldsymbol{\varepsilon}$ is learned by minimizing the following loss function,

$$\mathcal{L}_\varepsilon = -\beta \sum_{n=2}^{D} \left|\varepsilon_1 - \varepsilon_n\right|^2, \tag{5.5}$$

where β is a weight, and $\varepsilon_1, \ldots, \varepsilon_D$ are elements of the regularization vector $\boldsymbol{\varepsilon}$. Note that, to ensure the strict positive definiteness of $\boldsymbol{\varepsilon}$, we do not optimize it directly. Instead, we introduce an auxiliary vector $\hat{\boldsymbol{\varepsilon}}$ with components $\varepsilon_i = \hat{\varepsilon}_i^2 + 10^{-10}$, $i = 1, \ldots, D$. By squaring $\hat{\varepsilon}_i$ and adding this very small constant, we guarantee that every ε_i remains strictly positive. Thus, the overall loss is defined as $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{vel}} + \mathcal{L}_\varepsilon$. Figure 5.2–*b* shows a contractive dynamical system employing this method. The path integrals suggest that the system exhibits higher contraction when compared to the naive constant regularization shown in Fig. 5.2–*a*. This is quantitatively supported by the contraction measure statistics reported in Fig. 5.4, showing that the trajectories converge approximately 276% faster towards the data support. The reported 276%

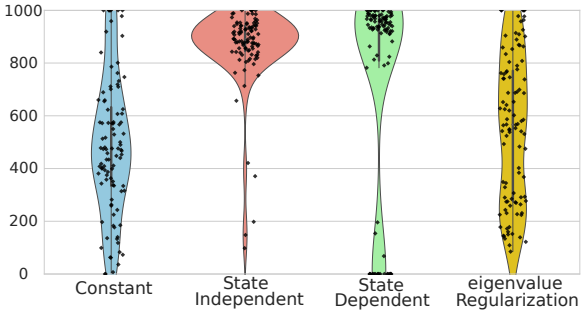


Figure 5.4: The distribution of the duration, in time steps, that each path integral remained within the demonstration region across four different regularization methods: *Constant*, *State Independent*, *State Dependent*, and *Eigenvalue Regularization*. A higher count of time steps inside the region indicates that trajectories likely converge faster and exhibit stronger contraction properties. Individual data points are represented by black dots.

faster convergence is based on the number of time steps each integral curve spends inside the demonstration region defined by the convex hull $\mathcal{H}$. This measure reflects how quickly trajectories enter and remain within the data support in comparison to vanilla NCDS. As shown in Table 5.1, this method achieves a higher contraction rate and ratio compared to vanilla NCDS.

State-dependent: In this approach, the regularization vector $\boldsymbol{\varepsilon} = g(\boldsymbol{x})$ is computed as a function of the state $\boldsymbol{x}$ using a neural network $g_\delta(\boldsymbol{x}) : \mathbb{R}^D \to \mathbb{R}^D$ with parameters $\boldsymbol{\delta}$. To ensure that $\boldsymbol{\varepsilon}$ remains strictly positive, we reparameterize it as $\varepsilon_i = \hat{\varepsilon}_i^2 + 10^{-10}$. Thus, the neural network g_δ directly outputs the auxiliary vector $\hat{\varepsilon}$, ensuring that the computed $\boldsymbol{\varepsilon}$ is always positive. The parameters $\boldsymbol{\delta}$ are learned via back propagation, which aims at minimizing the loss introduced in (5.5). Figure 5.2–*c* shows a contractive dynamical system using this state-dependent method, exhibiting faster trajectory convergence compared to the naive constant regularization (Fig. 5.2–*a*). Figure 5.4 shows that trajectories converge approximately 100% faster towards the data support. However, as indicated in Table 5.1, this method exhibits a lower contraction rate and ratio compared to vanilla NCDS. However, we noticed that the neural networks g_δ might produce very small regularization vectors, potentially generating spurious attractors.

5.2.2 Eigenvalue Regularization

Consider the Jacobian matrix $\hat{\boldsymbol{J}}_f$ associated with the dynamical system f, whose eigenvalue decomposition is given by $\hat{\boldsymbol{J}}_f = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{-1}$, where $\boldsymbol{V}$ is the matrix of eigenvectors of $\hat{\boldsymbol{J}}_f$, and $\boldsymbol{\Lambda}$ is the diagonal matrix of eigenvalues, $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_D)$. To incorporate regularization directly on the eigenvalues, we can use the same loss as in (5.5), with the difference that we manipulate the eigenvalues directly. For example, a simple loss encouraging the deviation of eigenvalues from a chosen eigenvalue λ_{i} is,

$$\mathcal{L}_\epsilon = -\beta \sum_{n=1}^{D} \left|\lambda_i - \lambda_n\right|^2, \tag{5.6}$$

where β is a weight, and the final loss includes both the velocity reconstruction loss and the regularization loss. Note that the main limitation of this approach is its computational cost due to the need to backpropagate through the eigenvalue decomposition. Figure 5.2–d illustrates the impact of this method on the system's contraction, showing relatively modest gains on the convergence of trajectories compared to the naive constant regularization approach depicted in Fig. 5.2–a. This is evident from the contraction measure statistics reported in Fig. 5.4, showing that trajectories converge only 13% faster towards the data support.

In conclusion, as regularization techniques become more complex, predicting their impact on the system's ability to generalize beyond the training data becomes increasingly challenging. The results in Fig. 5.2 suggest that the *vector-value state-independent* method provides a better contractive behavior toward the demonstration region and generalization outside of the data support. Also, the computational cost of this approach is lower than the alternatives as it does not rely on a neural network or an eigen-decomposition operation. Furthermore, Table 5.1 reports a comparison of the contraction rate and maximum contraction ratio, highlighting the superior performance of the state-independent method. More details on the evaluation of these eigenvalue metrics can be found in Sec. 6.4.3.

Notably, across every dataset and under the considered metrics, the state-independent method consistently outperformed the others. In our experiments, trajectories converged approximately 276% faster to the demonstration region, highlighting the improved performance in convergence and generalization. Based on these findings, we will henceforth adopt this regularization strategy as our default configuration for further analyses.

Next, we turn our attention to the other key component of (5.1), namely the symmetric matrix $\boldsymbol{J}_\theta(\boldsymbol{x})^\top \boldsymbol{J}_\theta(\boldsymbol{x})$. Our objective is to explore the effects of modifying the Jacobian from its symmetric form to an asymmetric one, with a particular focus on understanding how the introduction of a skew-symmetric component impacts the generalization behavior of the contractive dynamical system.

5.3 Asymmetry of the Jacobian

The NCDS builds on a symmetric Jacobian, as formulated in (5.1). However, it is important to note that for a dynamical system to be contractive, its Jacobian does not need to be symmetric (Jaffe et al., 2024). As Jaffe et al. (2024) discuss, a simple form of a contractive vector field with an asymmetric Jacobian can be expressed as the following linear system: $\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x}$ with $\boldsymbol{A} = \begin{pmatrix} -1 & 4 \\ 0 & -1 \end{pmatrix}$. The vector field produced by this system is shown in Fig. 5.3–*right*.

Table 5.2: The contraction rate and maximum contraction ratio computed over the equidistant grids.

Metrics	Contraction rate	Contraction ratio (max)
Angle (Sym.)	**−0.034**	**2.489**
Angle (Asym.)	−0.006	0.362
Sine (Sym.)	**−0.042**	**1.457**
Sine (Asym.)	−0.003	0.439

Therefore, we can enhance the NCDS flexibility by reformulating the system's Jacobian to incorporate both a symmetric and a skew-symmetric component. It is important to emphasize that adding the skew-symmetric component does not affect the negative definiteness of the symmetric part of the resulting Jacobian. This can be achieved by parameterizing the Jacobian using two separate neural networks: one generating the symmetric Jacobian, denoted as $\boldsymbol{J}_\theta$, and the other generating the skew-symmetric Jacobian, denoted as $\boldsymbol{J}_\phi$. The combined Jacobian is then

$$\hat{\boldsymbol{J}}_{\theta,\phi}(\boldsymbol{x}) = \hat{\boldsymbol{J}}_\theta(\boldsymbol{x}) + \hat{\boldsymbol{J}}_\phi(\boldsymbol{x}), \tag{5.7}$$

where the skew-symmetric matrix $\hat{\boldsymbol{J}}_\phi(\boldsymbol{x})$ is given by,

$$\hat{\boldsymbol{J}}_\phi(\boldsymbol{x}) = \begin{bmatrix} 0 & -\boldsymbol{J}_{\phi,0}(\boldsymbol{x}) & \boldsymbol{J}_{\phi,1}(\boldsymbol{x}) \\ \boldsymbol{J}_{\phi,0}(\boldsymbol{x}) & 0 & -\boldsymbol{J}_{\phi,2}(\boldsymbol{x}) \\ -\boldsymbol{J}_{\phi,1}(\boldsymbol{x}) & \boldsymbol{J}_{\phi,2}(\boldsymbol{x}) & 0 \end{bmatrix}. \tag{5.8}$$

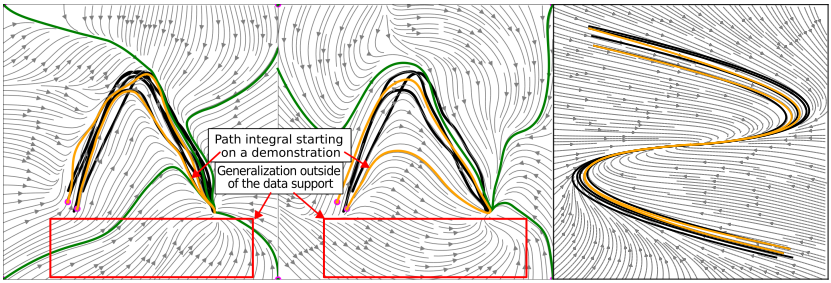


Figure 5.5: Comparison of 2D vector fields learned with different Jacobian formulations. *Left:* Vector field obtained using the symmetric Jacobian formulation. *Middle:* Vector field obtained using the asymmetric Jacobian formulation. The learned vector field (grey) and demonstrations (black). Yellow and green trajectories show path integrals starting from demonstration starting points and random points, respectively. *Right:* Vector field with asymmetric Jacobian learned using NCDS with symmetric Jacobian.

Here, $\boldsymbol{J}_{\boldsymbol{\phi}}(\boldsymbol{x})$ is parameterized by a neural network that outputs the three independent components $[\boldsymbol{J}_{\boldsymbol{\phi},0}(\boldsymbol{x}), \boldsymbol{J}_{\boldsymbol{\phi},1}(\boldsymbol{x}), \boldsymbol{J}_{\boldsymbol{\phi},2}(\boldsymbol{x})]$ of the skew-symmetric matrix. Note that this formulation is tailored for the three-dimensional case and does not readily generalize to higher dimensions. While this reformulation can enhance the expressiveness and flexibility of NCDS, it is essential to assess its impact on the generalization behavior of the learned vector field. Specifically, Fig. 5.5–*middle* illustrates the behavior of a learned contractive dynamical system with an asymmetric Jacobian, $\hat{\boldsymbol{J}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x})$, as described in (5.7). This is compared to the learned contractive vector field shown in Fig. 5.5–*left*, which has only a symmetric Jacobian, as described in (5.1). In these plots, the arrows indicate path integrals that originate from one of the initial points in the demonstrations. For the system with an asymmetric Jacobian (Fig. 5.5–*middle*), the path integral diverges from the data support, effectively taking a shortcut. In contrast, the path integral for the system with a symmetric Jacobian (Fig. 5.5–*left*) closely follows the data trend and remains within the data region. The red rectangles emphasize an area far from the data, requiring network generalization. In the asymmetric Jacobian case (Fig. 5.5–*middle*), the system initially diverges the path integrals before redirecting them back to the target. Conversely, the system with the symmetric Jacobian (Fig. 5.5–*left*) guides the path integrals directly towards the data region without diverging first. To further analyze the contraction and generalization behavior of NCDS under both conditions, we computed multiple path integrals starting from an equidistant grid around the demonstration trajectories.

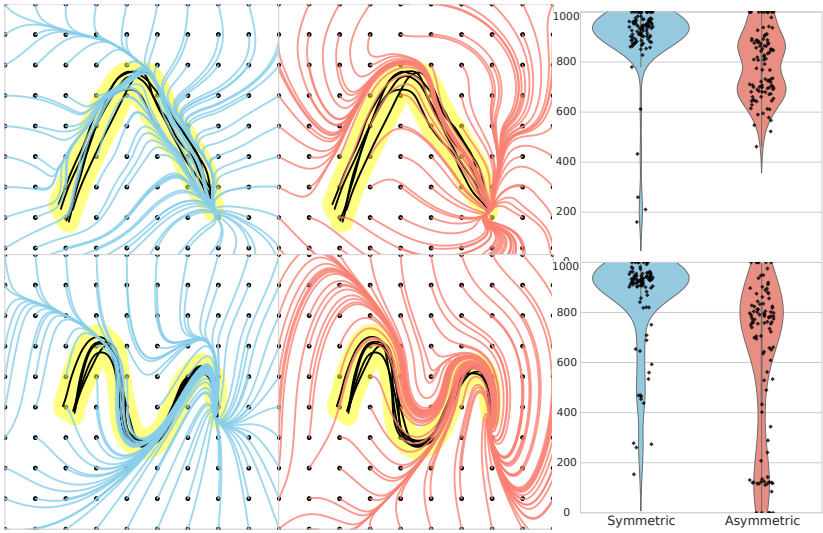


Figure 5.6: Comparison between asymmetric and symmetric Jacobians in learning contractive dynamical systems. *Left:* Generalization behavior of a system with a symmetric Jacobian, visualized using path integrals originating from a 10×10 equidistant grid. The yellow region represents the demonstration area, with black curves denoting the demonstrations. Black dots indicate the initial points of the path integrals, while blue curves illustrate the resulting trajectories. *Middle:* Generalization behavior of a system with an asymmetric Jacobian. Pink curves represent the path integrals. *Right:* Comparison of the number of frames each path integral spent within the demonstration region. A higher number indicates a path integral more likely converged to the demonstrations, therefore, better contractive behavior.

As shown in Fig. 5.6–*left* and 5.6–*middle*, the yellow region defines the area where the demonstrations, depicted as black curves, reside. Figures 5.6–*left* and 5.6–*middle* display all the path integrals generated by NCDS with symmetric and asymmetric Jacobians, respectively. Figure 5.6–*right* shows the number of time steps spent inside the demonstration region. As illustrated, the path integrals with symmetric Jacobians spend more time within this region, i.e. the trajectories reached the data support faster, suggesting a more effective contractive behavior. Additionally, as confirmed by the results in Table 5.2, both the contraction ratio and contraction rate for the *Angle* and *Sine* datasets exhibit improved contractive behavior, indicating that the learned dynamics are more contractive under symmetric Jacobian. More details on the evaluation of these eigenvalue metrics can be found in Sec. 6.4.4. Notice that all results in Fig. 5.5, Fig. 5.6

(and Fig. 6.12 in appendix) are obtained using the *state-independent regularization vector*, with the only distinction being the symmetry properties of the Jacobian matrix.

Our formulation employs a local approximation paradigm where the network estimates a distinct Jacobian at each point in state space, rather than assuming a "universal" Jacobian for the entire space. This local tailoring proves experimentally sufficient for approximating contractive vector fields, even when using a symmetric Jacobian. Although a non-symmetric Jacobian may offer greater theoretical expressivity, our empirical results suggest that the symmetric Jacobian exhibits superior generalization properties, making it the preferred choice in practice. This is shown in Fig. 5.5–*right*, where NCDS with a symmetric Jacobian successfully learns the dynamics of an asymmetric system, shown in Fig. 5.3–*right*.

In conclusion, although a model with an asymmetric Jacobian could theoretically learn a broader range of contractive dynamical systems, our preliminary results presented in Fig. 5.5 and Fig. 5.6 did not show any significant improvement in reconstruction accuracy or generalization performance. Moreover, empirical investigations presented in the results sections, including both the LASA dataset (Sec. 5.6.1) and the robotic and human motion results (Sec. 6.4.7), have not identified any dynamical systems where the symmetric Jacobian approach failed to effectively capture and learn the underlying dynamics.

In the next section, we will focus on how to equip NCDS with obstacle avoidance capabilities using modulation matrices.

5.4 Obstacle Avoidance via Matrix Modulation

In this section, we review the matrix modulation technique employed by NCDS for obstacle avoidance. Subsequently, in Sec. 6.3, we will explore how this method can be extended using Riemannian manifold learning to navigate obstacles and avoid unsafe regions. In a dynamic environment with obstacles, a learned contractive dynamical system should effectively adapt to and avoid unseen obstacles, without interfering with the global contracting behavior of the system. Vanilla NCDS is equipped with a contraction-preserving obstacle avoidance technique that builds on the dynamic modulation matrix $\boldsymbol{G}$ introduced by (Huber et al., 2022). This approach locally reshapes the learned vector field in the proximity of obstacles, while preserving contraction properties within the forward invariant safe set. A set $\mathcal{S} \subseteq \mathbb{R}^D$ is forward invariant for the dynamical system $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$ if, for every initial condition $\boldsymbol{x}_0 \in \mathcal{S}$, the solution

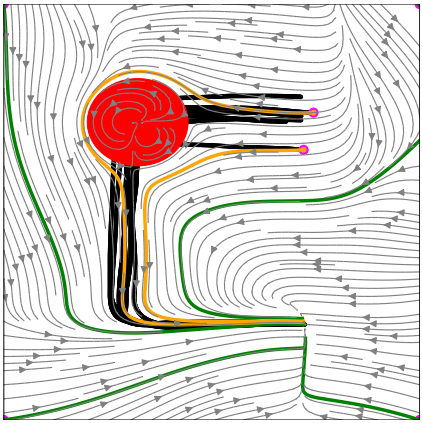


Figure 5.7: The obstacle locally reshapes the learned vector field using the modulation matrix. The gray contours represent the learned vector field, black trajectories depict the demonstrations, and orange/green trajectories are path integrals starting from both initial points of the demonstrations and plot corners. Magenta and red circles indicate the initial points of the path integrals and the obstacle, correspondingly.

satisfies $\boldsymbol{x}_t \in \mathcal{S}$ for all $t \geq 0$. In our context, $\mathcal{S}$ is defined as the obstacle-free region, i.e., $\mathcal{S} \cap \mathcal{O} = \emptyset$, where $\mathcal{O}$ denotes the set of points occupied by obstacles.

Specifically, Huber et al. (2022) show how to construct a modulation matrix $\boldsymbol{G}$ from the obstacle's location and geometry, such that the vector field $\dot{\boldsymbol{x}} = \boldsymbol{G}(\boldsymbol{x}) f_\theta(\boldsymbol{x})$ is both contractive and steers around the obstacle. Formally, given the modulation matrix $\boldsymbol{G}$, we can reshape the vector field

$$\hat{\dot{\boldsymbol{x}}} = \boldsymbol{G}(\boldsymbol{x}) f_\theta(\boldsymbol{x}), \tag{5.9}$$

$$\boldsymbol{G}(\boldsymbol{x}) = \boldsymbol{E}(\boldsymbol{x}) \boldsymbol{D}(\boldsymbol{x}) \boldsymbol{E}(\boldsymbol{x})^{-1}, \tag{5.10}$$

where $\boldsymbol{E}(\boldsymbol{x})$ and $\boldsymbol{D}(\boldsymbol{x})$ are the basis and diagonal eigenvalue matrices computed as,

$$\boldsymbol{E}(\boldsymbol{x}) = [\boldsymbol{n}(\boldsymbol{x})\ \mathbf{e}_1(\boldsymbol{x}) \dots \mathbf{e}_{d-1}(\boldsymbol{x})], \tag{5.11}$$

$$\boldsymbol{D}(\boldsymbol{x}) = \mathrm{diag}(\lambda_n(\boldsymbol{x}) \lambda_\tau(\boldsymbol{x}), \dots, \lambda_\tau(\boldsymbol{x})), \tag{5.12}$$

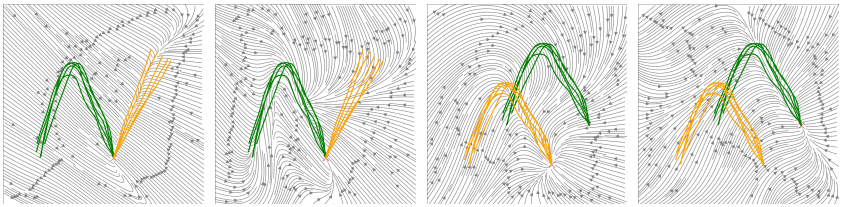
where $\boldsymbol{n}(\boldsymbol{x}) = \frac{\boldsymbol{x} - \boldsymbol{x}_r}{\|\boldsymbol{x} - \boldsymbol{x}_r\|}$ is a reference direction computed w.r.t. a reference point $\boldsymbol{x}_r$ on the obstacle, and the tangent vectors $\mathbf{e}_i$ form an orthonormal basis to the gradient of the distance function $\Gamma(\boldsymbol{x})$ (see Huber et al. (2022) for its full derivation). Moreover, the

components of the matrix $\boldsymbol{D}$ are defined as $\lambda_n(\boldsymbol{x}) = 1 - \left(\frac{1}{\Gamma(\boldsymbol{x})}\right)^{\frac{1}{\rho}}$, $\lambda_\tau(\boldsymbol{x}) = 1 + \left(\frac{1}{\Gamma(\boldsymbol{x})}\right)^{\frac{1}{\rho}}$, where $\rho \in \mathbb{R}^+$ is a reactivity factor. Note that the matrix $\boldsymbol{D}$ modulates the dynamics along the directions of the basis defined by the set of vectors $\boldsymbol{n}(\boldsymbol{x})$ and $\mathbf{e}(\boldsymbol{x})$. As stated by Huber et al. (2022), the function $\Gamma(\cdot)$ monotonically increases w.r.t the distance from the obstacle's reference point $\boldsymbol{x}_r$, and it is, at least, a C^1 function. Importantly, the modulated dynamical system $\hat{\dot{\boldsymbol{x}}} = \boldsymbol{G}(\boldsymbol{x}) f_\theta(\boldsymbol{x})$ still guarantees contractive stability, which can be proved by following the same proof provided by (Huber et al., 2019). Figure 5.7 shows the application of a modulation matrix to navigate around an obstacle in a toy example. The obstacle, depicted as a red circle, completely obstructs the demonstrations. Notably, NCDS successfully generated safe trajectories by effectively avoiding the obstacle and ultimately reaching the target.

Having completed our discussion on improving the Jacobian formulation in (5.1), we now shift focus to investigating the ability of a single NCDS module to learn and represent multiple contractive vector fields, each corresponding to a conditional value such as trajectory targets or shapes.

5.5 Conditional NCDS

Although NCDS has the ability to generate contractive vector fields for executing complex skills, it lacks the ability to handle multiple motion skills, which may be achieved by conditioning on task-related variables such as target states. In this context, other methods that leverage contraction stability often depend heavily on rigid optimization processes, which limits their adaptability when confronted with varying conditional values. These methods typically require either finding a new contraction metric each time the condition changes—likely by considering new constraints for optimization (Tsukamoto et al., 2021c)—or, in other cases, generating a contraction metric for every condition when using Neural Contraction Metrics (NCMs) (Tsukamoto et al., 2021b). Although Jaffe et al. (2024) introduced a contraction method that dynamically adapts to a varying target, due to the extended linearization used to parametrize the vector field, it does not consider other types of conditioning task variables, e.g., variations in trajectory shape or switching between multiple target points. Here, we present the concept of conditional NCDS (CNCDS), which extends the NCDS with the ability of learning multimodal tasks, which depend on context variables such as variable targets, using a single NCDS module.



Reproduce *angle* trajectory | Reproduce *line* trajectory | Reach point A | Reach point B

Figure 5.8: CNCDS on 2D trajectories from the LASA handwriting dataset. Plots (a) and (b) show CNCDS conditioned on the trajectory shape. Plots (c) and (d) display conditioning on the trajectory target, where the conditional value is chosen to be 0 and 1, respectively.

The conditional variable can account for changes in the task conditions and further expand the NCDS architecture to integrate perception systems such as a vision perception backbone. To do so, first we introduce a new condition variable $\boldsymbol{\varpi}$ in the formulation. This variable is concatenated to the state vector $\boldsymbol{x}$ so that CNCDS retrieves a velocity vector $\dot{\boldsymbol{x}}$ as a function of the state and the task condition. Therefore, we can reformulate (5.1) as,

$$\hat{\boldsymbol{J}}_f([\boldsymbol{x}, \boldsymbol{\varpi}]) = -\left(\boldsymbol{J}_{\boldsymbol{\theta}}([\boldsymbol{x}, \boldsymbol{\varpi}])^{\mathsf{T}} \boldsymbol{J}_{\boldsymbol{\theta}}([\boldsymbol{x}, \boldsymbol{\varpi}]) + \mathrm{diag}(\boldsymbol{\varepsilon})\right). \tag{5.13}$$

Figure 5.8 illustrates the vector field generated by a CNCDS trained under two distinct conditional settings: In the first, referred to as *shape conditioning*, the conditional values are 0.0 for angle motion and 1.0 for line motion. In the second setting, referred to as *target conditioning*, the condition variable $\boldsymbol{\varpi} = [x^*, y^*]$ represents the 2D coordinates of the target.

Figure 5.8–a and Fig. 5.8–b display the vector fields resulting from conditioning on the trajectory shape. Figure 5.8–a shows the vector field when the conditional vector $\boldsymbol{\varpi}$ leads the NCDS model to reconstruct motion characterized by the *angle* motion, whereas panel Fig. 5.8–b displays the vector field for the *line* motion. Furthermore, Fig. 5.8–c and Fig. 5.8–d show the vector fields conditioned on the trajectory target. The reported proof-of-concept experiments confirm that conditional variables can be effectively incorporated into our model, allowing a single trained CNCDS to generate motions of different shapes based on varying conditioning inputs. In Sec. 6.4.8, we show how a vision backbone can be used to design CNCDS for image-based applications. This process is visualized in the block C of the architecture in Fig. 6.1. Having introduced

all the components of NCDS, we now turn our attention to evaluating this approach in learning contractive vector fields across various experimental settings.

5.6 Experiments

In this section, we perform a preliminary evaluation of NCDS, focusing on learning low-dimensional vector fields and examining the improvements introduced by regularization. A more thorough and comprehensive testing of NCDS is conducted later in Sec. 6.4, where we evaluate its performance on various synthetic and real-world tasks.

5.6.1 Learning Contractive Vector Fields

There are currently no established benchmarks for contraction-stable robot motion learning, so we focus on the LASA dataset (Lemme et al., 2015), often used to benchmark asymptotic stability. This consists of 26 different two-dimensional hand-written trajectories, which the system is tasked to follow. To evaluate our method, we have selected 5 different letter shapes from this dataset. To ensure consistency and comparability, we preprocess all trajectories to stop at the same target state. Additionally, we omit the initial few points of each demonstration, to ensure that the only state exhibiting zero velocity is the target state.

We first evaluate NCDS on two-dimensional trajectories for ease of visualization. Here data are sufficiently low-dimensional that we do not consider a latent structure. The Jacobian network was implemented as a neural network with two hidden layers, each containing 500 nodes. The network's output was reshaped into a square matrix format. For integration, we used the efficient `odeint` function from the `torchdiffeq` Python package (Chen, 2018), which supports various numerical integration methods. In our experiments, we employed the widely used Runge-Kutta and `dopri5` methods for solving ordinary differential equations.

Figure 5.9 shows the learned vector fields (gray contours) for 5 different trajectory shapes chosen according to their difficulty from the LASA dataset, covering a wide range of demonstration patterns (black curves) and dynamics. The top row shows the behavior of the learned dynamics for Vanilla NCDS, while the bottom row depicts the dynamics learned with state-independent vector regularization (which showed the strongest contraction behavior as reported in Fig. 5.4). We observe that both NCDS approaches effectively capture and replicate the underlying dynamics. This is observed

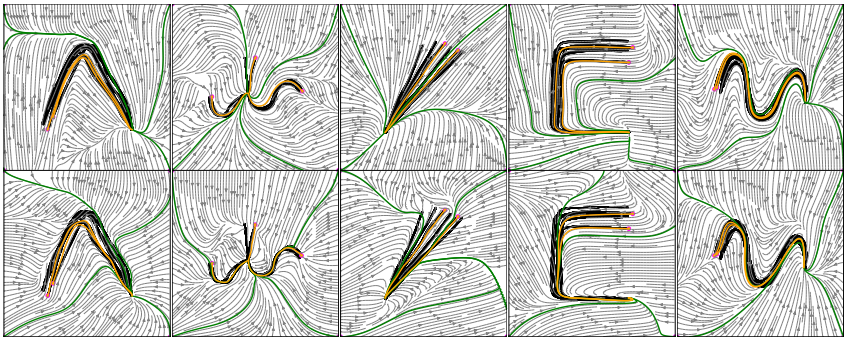


Figure 5.9: Visualization of the LASA-2D dataset: Gray contours represent the learned vector field, black trajectories depict demonstrations, and orange/green trajectories illustrate path integrals starting from the initial points of the demonstrations and plot corners. The magenta circles indicate the initial points of the path integrals.

in the orange path integrals, starting from the initial points of the demonstrations, showing that both approaches can reproduce the demonstrated trajectories accurately.

Table 5.3: Average time steps spent outside the data region for 100 path integrals, each consisting of 1000 points, originating from an equidistant grid (lower is better). The datasets and NCDS models correspond to those displayed in Fig. 5.9.

Dataset	Angle	Multimodels	Line	SharpC	Sine
Constant Regularization	516	436	445	490	439
State-independent Regularization Vector	135	105	255	209	176

We compute additional path integrals starting from outside the data support (green curves) to assess the generalization capability of both NCDS approaches beyond the observed demonstrations. Table 5.3 lists the average number of time steps that 100 path integrals, each consisting of 1000 points and originating from an equidistant grid, spent outside the data region for the examples shown in Fig. 5.9. Low values indicate that the trajectories converge quicker toward the demonstration region, reflecting improved contractive behavior. These results suggest that state-independent vector regularization enhance the contractive behavior (see also Fig. 5.4). This is evidence that the contractive construction is a viable approach to controlling the extrapolation properties of the neural network.

5.6.2 Ablation Studies

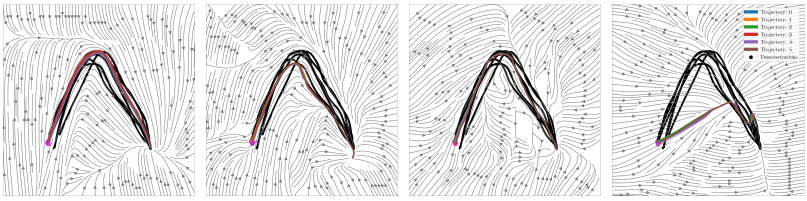
In this section, we perform ablation studies to analyze the impact of various components of our NCDS framework. By systematically altering or removing key elements, we aim to highlight the significance of the contraction constraint, the choice of network architecture, and the effect of different activation functions.

Unconstrained dynamical system

To illustrate the influence of having a negative definite Jacobian in NCDS, we contrast our NCDS against an identical system, except that we remove the negative-definiteness constraint on the Jacobian $\hat{\boldsymbol{J}}_{\theta}$. Figure 5.11 shows the path integrals generated by both contractive (Fig. 5.11-a) and unconstrained (Fig. 5.11-b) dynamical systems. As anticipated, the dynamics generated by the unconstrained system lack stable behavior. However, the vector field aligns with the data trends in the data support regions. Furthermore, we performed similar experiments with two different baseline approaches: an MLP and a Neural Ordinary Differential Equation (NeuralODE) network, to highlight the effect of the contraction constraint on the behavior of the dynamical system. The MLP baseline uses a neural network with 2 hidden layers each with 100 neurons with Tanh activation function. The NeuralODE baseline is implemented based on the code provided by Poli et al. (2021), with 2 hidden layers each with 100 neurons. The model is then integrated into a NeuralODE framework configured with the adjoint sensitivity method and the dopri5 solver for both the forward and adjoint passes. It is worth mentioning that these baselines directly reproduce the velocity according to $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$. Both networks are trained for 1000 epochs with the ADAM (Kingma et al., 2014a). Figure 5.11-c and 5.11-d show that both models effectively capture the observed dynamics (vector field); however, as anticipated, contraction stability is only achieved by NCDS.

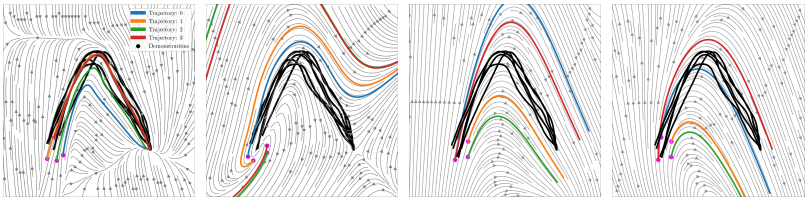
Activation function

The choice of activation function certainly affects the generalization in the dynamical system in areas outside the data support. To show this phenomena, we ablate a few common activation functions. For this experiment, we employ a feedforward neural network with two hidden layers, each of 100 units. As shown in Fig. 5.12, both the Tanh and Softplus activation functions give superior performance, coupled with satisfactory



$\epsilon = 10^{-4}$ $\epsilon = 10^{-2}$ $\epsilon = 10^{-1}$ $\epsilon = 10$

Figure 5.10: Path integrals generated by NCDS trained using different regularization term ϵ.



Contractive Unconstrained MLP Neural ODE

Figure 5.11: Path integrals generated under the Neural Contractive Dynamical Systems (NCDS) setting, along with baseline comparisons using Multilayer Perceptron (MLP) and Neural Ordinary Differential Equation (NeuralODE) models.

generalization capabilities. This indicates that the contour of the vector field aligns with the overall demonstration behavior outside of the data support. Conversely, the Sigmoid activation function yields commendable generalization beyond the confines of the data support, but it fails to reach and stop at its target.

Zero contraction ratio vector fields:

In our formulation, the regularization loss terms serve as soft constraints that guide the learning process without imposing a rigid structure on the solution. For example, when the ground-truth vector field (see Fig. 5.13, left panel) exhibits straight-line trajectories converging to the origin, the network prioritizes an accurate reconstruction of such demonstrated behavior. This is achieved without enforcing a large disparity in the regularization vector components or the eigenvalues spread. To illustrate this

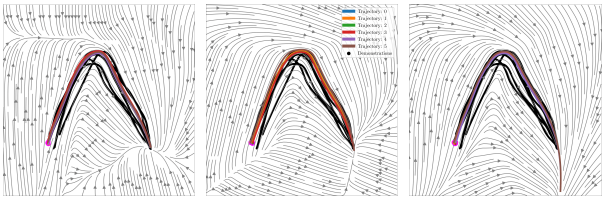


Figure 5.12: Path integrals generated by NCDS trained using different activation functions.

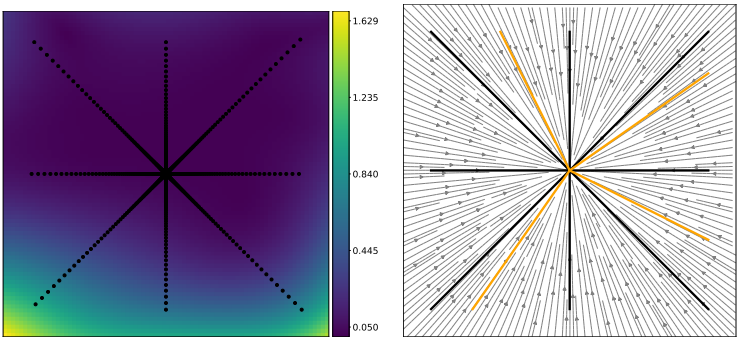


Figure 5.13: Learned vector field with zero contraction ratio. *Left:* The contraction ratio, scaled by 1000, highlights minimal eigenvalue differences. *Right:* Orange integral curves closely follow the training demonstrations (black), with gray contours outlining the learned vector field.

phenomenon, Fig. 5.13 shows a comparative analysis. In the right panel, the black trajectories represent the training demonstrations, while the orange trajectories are the integral curves generated by the model. The results indicate that the model effectively captures the linear behavior of the ground-truth vector field. Additionally, the scaled contraction ratio (i.e., $\times 1000$) demonstrates that the differences between the eigenvalues are effectively negligible.

In conclusion, the results above show that NCDS can learn stable vector fields that generalize reliably beyond the data-support regions and avoid obstacles while preserving contraction; however, scalability to high-dimensional observations and orientation states remains a challenge because enforcing contraction and safety requires expensive Jacobian/metric regularization that scales with dimension. The next chapter addresses

this by learning contractive dynamics in a low-dimensional latent space and using an injective decoder so that contraction is preserved when mapping back to the data space. We then extend the formulation to orientation dynamics on Lie groups and introduce Riemannian safety regions that enable efficient obstacle avoidance directly in the latent space.

6 Latent Contractive Dynamics

*This chapter is based on the paper **Neural Contractive Dynamical Systems**, published in International Conference on Robot Learning (ICLR) in 2024 (Beik-Mohammadi et al., 2024), and the paper **Extended Neural Contractive Dynamical Systems: On Multiple Tasks and Riemannian Safety Regions**, published in International Journal on Robotics Research in 2024 (Beik-Mohammadi et al., 2025).*

Learning highly nonlinear contractive dynamical systems in high-dimensional spaces is difficult. These systems may exhibit complex trajectories with intricate interdependencies among the system variables, making it challenging to capture the underlying dynamics while ensuring a contractive behavior. In the proof-of-concept examples previously discussed, we showed that NCDS works very well for low-dimensional problems, but when only limited data is available, the approach becomes brittle in higher dimensions. A common approach in such cases is to first reduce the data dimensionality, and work as before in the resulting low-dimensional latent space (Chen et al., 2018b, Hung et al., 2022, Beik-Mohammadi et al., 2021). The main challenge is that even if the latent dynamics are contractive, the associated high-dimensional dynamics need not be. This we solve next. To begin with, we review the necessary background concepts.

6.1 Latent NCDS

To obtain a low-dimensional representation of the demonstration data, the NCDS leverages Variational Autoencoders (VAEs) to encode high-dimensional vector fields into a low-dimensional latent space. Therefore, we will review the background on VAEs. However, while most functionalities of NCDS can be effectively transferred to the latent space, obstacle avoidance remains an exception. Therefore, this specific task still needs to be performed in the original high-dimensional space, where the computational complexity can negatively impact the system performance. To overcome this, Sec. 6.3 introduces an approach that equips NCDS with an alternative modulation formulation,

enabling the transfer of this computationally-intensive obstacle avoidance process to the latent space, thus improving overall efficiency. We will later demonstrate that this is achieved by leveraging VAEs and their ability to learn Riemannian manifolds, which represent the underlying geometrical structure of the data in the latent space through pullback metrics.

As briefly discussed above, we want to reduce the data dimensionality with a VAE, but we further require that any latent contractive dynamical system remains contractive after it has been decoded into the data space. To do so, we leverage the fact that contraction is invariant under coordinate changes (Manchester et al., 2017, Kozachkov et al., 2023). This means that the transformation between the latent and data spaces may be generally achieved through a diffeomorphic mapping.

Theorem 6.1 (Contraction invariance under diffeomorphisms (Manchester et al., 2017))**.** *Given a contractive dynamical system $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$ and a diffeomorphism ψ applied on the state $\boldsymbol{x} \in \mathbb{R}^D$, the transformed system preserves contraction under the change of coordinates $\boldsymbol{y} = \psi(\boldsymbol{x})$. Equivalently, contraction is also guaranteed under a differential coordinate change $\delta_{\boldsymbol{y}} = \frac{\partial \psi}{\partial \boldsymbol{x}} \delta_{\boldsymbol{x}}$.*

Following Theorem 6.1, we learn a VAE with a smooth injective decoder $\mu : \mathcal{Z} \to \mathcal{X}$. Letting $\mathcal{M} = \mu(\mathcal{Z})$ denote the image of μ, then μ is a diffeomorphism between $\mathcal{Z}$ and $\mathcal{M}$, such that Theorem 6.1 applies. Geometrically, μ spans a d-dimensional submanifold of $\mathcal{X}$ on which the dynamical system operates.

Here we leverage the zero-padding architecture from Brehmer et al. (2020) for the decoder. Formally, an injective flow $\mu : \mathcal{Z} \to \mathcal{X}$ learns an injective mapping between a low-dimensional latent space $\mathcal{Z}$ and a higher-dimensional data space $\mathcal{X}$. Injectivity of the flow ensures that there are no singular points or self-intersections in the flow, which may compromise the stability of the system dynamics in the data space. The injective decoder μ is composed of a zero-padding operation on the latent variables followed by a series of K invertible transformations ι_k. This means that,

$$\mu = \iota_K \circ \cdots \circ \iota_1 \circ \text{Pad}, \tag{6.1}$$

where $\text{Pad}(\boldsymbol{z}) = \begin{bmatrix} z_1 & \cdots & z_d & 0 & \cdots & 0 \end{bmatrix}^\top$ represents a D-dimensional vector $\boldsymbol{z}$ with additional $D - d$ zeros, and $\circ$ denotes function decomposition. We emphasize that this decoder is an injective mapping between $\mathcal{Z}$ and $\mu(\mathcal{Z}) \subset \mathcal{X}$, such that a decoded contractive dynamical system remains contractive.

Specifically, we propose to learn a latent data representation using a VAE, where the decoder mean μ_ξ follows the architecture in (6.1). Empirically, we have found that training stabilizes when the variational encoder takes the form $q_\xi(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z} \mid \mu_\xi^{\sim 1}(\boldsymbol{x}), \mathbb{I}_d \sigma_\xi^2(\boldsymbol{x}))$, where $\mu_\xi^{\sim 1}$ is the approximate inverse of μ_ξ given by,

$$\mu_\xi^{\sim 1} = \text{Unpad} \circ \imath_1^{-1} \circ \cdots \circ \imath_K^{-1}, \tag{6.2}$$

where $\text{Unpad} : \mathbb{R}^D \to \mathbb{R}^d$ removes the last $D-d$ dimensions of its input as an approximation to the inverse of the zero-padding operation. We emphasize that an exact inverse is not required to evaluate a lower bound of the model evidence. This process is visualized in block A of the architecture in Fig. 6.1.

It is important to note that the state $\boldsymbol{x}$ solely encodes the positional information of the system, disregarding the velocity $\dot{\boldsymbol{x}}$. In order to decode the latent velocity $\dot{\boldsymbol{z}}$ into the data space velocity $\dot{\boldsymbol{x}}$, we exploit the Jacobian matrix associated with the decoder mean function μ_ξ, computed as $\boldsymbol{J}_{\mu_\xi}(\boldsymbol{z}) = {}^{\partial \mu_\xi}/_{\partial z}$. This enables the decoding process formulated as below,

$$\dot{\boldsymbol{x}} = \boldsymbol{J}_{\mu_\xi}(\boldsymbol{z})\dot{\boldsymbol{z}}. \tag{6.3}$$

The above tools let us learn a contractive dynamical system on the latent space $\mathcal{Z}$, where the contraction is guaranteed by employing the NCDS architecture (Sec. 5.1). Then, the latent velocities $\dot{\boldsymbol{z}}$ given by such a contractive dynamical system can be mapped to the data space $\mathcal{X}$ using (6.3). For training, the latent velocities are simply estimated by a numerical differentiation w.r.t the latent state $\boldsymbol{z}$. Assuming the initial robot configuration $\boldsymbol{x}_0$ is in $\mathcal{M}$ (i.e., $\boldsymbol{x}_0 = f(\boldsymbol{z}_0)$), the subsequent motion follows a contractive system along the manifold. If the initial configuration $\boldsymbol{x}_0$ is not in $\mathcal{M}$, the encoder is used to approximate a projection onto $\mathcal{M}$, producing $\boldsymbol{z}_0 = \mu^{\sim 1}(\boldsymbol{x}_0)$.

Although the transition from $\boldsymbol{x}_0$ to $\mu(\boldsymbol{z}_0)$ may not exhibit contractive behavior, this phase is of finite duration; once the state is on $\mathcal{M}$, the contractive dynamics ensure global stability on the decoder's manifold, leading to convergence of the overall system. This behavior can be observed in high-dimensional data, where initial deviations may occur. However, the contractive dynamics on the manifold guarantee stability, as explained in the example given in Sec. 6.4.2.

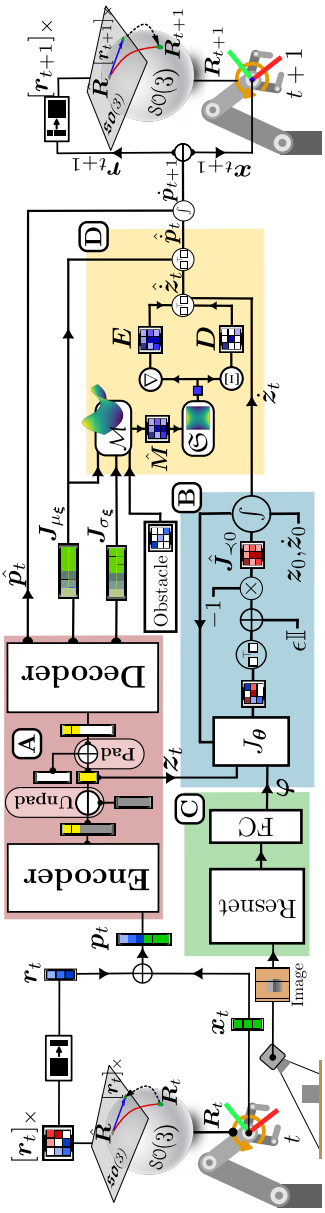


Figure 6.1: Architecture overview: a single iteration of NCDS simultaneously generating position and orientation dynamics. **(A) VAE (pink box):** The encoder processes the concatenated position-orientation data $\boldsymbol{p}_t$, yielding a resulting vector that is subsequently divided into two components: the latent code $\boldsymbol{z}$ (yellow squares) and the surplus (gray squares). The Unpad function in (6.2) removes the unused segment. The unpadded latent code $\boldsymbol{z}$ is fed to the contraction module and simultaneously padded with zeros (white squares) before being passed to the injective decoder. **(B) Contraction (blue box):** The Jacobian network output, given the latent codes, is reshaped into a square matrix and transformed into a negative definite matrix using (2.15). The numerical integral solver then computes the latent velocity $\dot{\boldsymbol{z}}$. Later, using (6.3), $\dot{\boldsymbol{z}}$ is mapped to the input-space velocity via the decoder's Jacobian $\boldsymbol{J}_{\boldsymbol{\mu}_\xi}$. **(C) Vision backbone (green box):** The vision backbone enhances the model by allowing it to adapt to different task conditions through visual perception. This component processes visual inputs, which are then concatenated with the state vector $\boldsymbol{x}$, enabling the CNCDS to generate task-specific dynamics. **(D) Riemannian modulation (yellow box):** Uses learned Riemannian manifolds in the VAE's latent space to implicitly represent obstacles, dynamically reshaping the vector field via a modulation matrix $\boldsymbol{G}_{\mathcal{M}}(\boldsymbol{x})$.

6.2 Learning Position and Orientation Dynamics

So far, we have focused on Euclidean robot states, but in practice, the end-effector motion also involves rotations, which do not have an Euclidean structure. We first review the group structure of rotation matrices and then extend NCDS to handle non-Euclidean data using Theorem 6.1.

6.2.1 Orientation Parameterization

Three-dimensional spatial orientations can be represented in several ways, including Euler angles, unit quaternions, and rotation matrices (Shuster, 1993). We focus on the latter approaches.

Rotation matrices $\mathcal{SO}(3)$

The set of rotation matrices forms a Lie group, known as the *special orthogonal group* $\mathcal{SO}(3) = \left\{ \boldsymbol{R} \in \mathbb{R}^{3\times3} \,\middle|\, \boldsymbol{R}^\mathsf{T}\boldsymbol{R} = \mathbb{I}, \det(\boldsymbol{R}) = 1 \right\}$. Every Lie group is associated with its Lie algebra, which represents the tangent space at its origin (see Fig. 6.2–*left*). This Euclidean tangent space allows us to operate with elements of the group via their projections on the Lie algebra (Solà et al., 2018). In the context of $\mathcal{SO}(3)$, its Lie algebra $\mathfrak{so}(3)$ is the set of all 3×3 skew-symmetric matrices $[\boldsymbol{r}]_\times$. This skew-symmetric matrix exhibits three degrees of freedom, which can be reparameterized as a 3-dimensional vector $\boldsymbol{r} = [r_x, r_y, r_z] \in \mathbb{R}^3$.

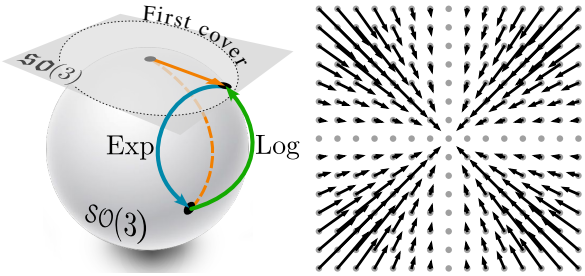


Figure 6.2: *Left*: Aspects of the Lie group $\mathcal{SO}(3)$. *Right*: An illustration of the function $b(\boldsymbol{x})$ in two dimensions.

We can map back and forth between the Lie group $\mathcal{SO}(3)$ and its associated Lie algebra $\mathfrak{so}(3)$ using the *logarithmic* and *exponential maps*, denoted $\text{Log} : \mathcal{SO}(3) \to \mathfrak{so}(3)$ and $\text{Exp} : \mathfrak{so}(3) \to \mathcal{SO}(3)$, which are defined as follows,

$$\text{Exp}([\boldsymbol{r}]_\times) = \mathbb{I} + \frac{\sin(\zeta)}{\zeta}[\boldsymbol{r}]_\times + \frac{1-\cos(\zeta)}{\zeta^2}[\boldsymbol{r}]_\times^2,$$
$$\text{Log}(\boldsymbol{R}) = \zeta \frac{\boldsymbol{R} - \boldsymbol{R}^\top}{2\sin(\zeta)},$$

where $\zeta = \arccos\left(\frac{\text{Tr}(\boldsymbol{R})-1}{2}\right)$. Moreover, $\boldsymbol{R}$ represents the rotation matrix, and $[\boldsymbol{r}]_\times$ denotes the skew-symmetric matrix associated with the coefficient vector $\boldsymbol{r}$. Due to wrapping (i.e., 360° rotation corresponds to 0°), the exponential map is surjective. This implies that the inverse, i.e. Log, is multivalued, which complicates matters. However, for vectors $\boldsymbol{r} \in \mathfrak{so}(3)$, both Exp and Log are diffeomorphic if $\|\boldsymbol{r}\| < \pi$ (Falorsi et al., 2019, Urain et al., 2022). This π-ball $\mathcal{B}_\pi$ is known as the *first cover* of the Lie algebra and corresponds to the part where no wrapping occurs.

Quaternions $\mathcal{S}^3$

Quaternions offer an alternative representation of rotations in 3D space and can also be endowed with a Lie group structure, leading to the *unit quaternion group* $\mathcal{S}^3 = \left\{\boldsymbol{q} \in \mathcal{S}^3 \subset \mathbb{R}^4 \,\middle|\, \|\boldsymbol{q}\| = 1\right\}$. Like rotation matrices, the unit quaternion group $\mathcal{S}^3$ is associated with a Lie algebra $\mathfrak{su}(2)$, which represents the tangent space at its origin and corresponds to the set of pure imaginary quaternions, which can be represented as 3D vectors.

A quaternion $\boldsymbol{q} \in \mathcal{S}^3$ is typically expressed as $\boldsymbol{q} = q_0 + q_x\boldsymbol{i} + q_y\boldsymbol{j} + q_z\boldsymbol{k}$, where q_0 is the scalar part and (q_x, q_y, q_z) represents the vector part. The exponential and logarithmic maps, denoted as $\text{Exp} : \mathfrak{su}(2) \to \mathcal{S}^3$ and $\text{Log} : \mathcal{S}^3 \to \mathfrak{su}(2)$, provide a way to transition between the Lie group and its algebra. These maps are defined as follows,

$$\text{Exp}(\boldsymbol{v}) = \cos\left(\frac{\|\boldsymbol{v}\|}{2}\right) + \sin\left(\frac{\|\boldsymbol{v}\|}{2}\right)\frac{\boldsymbol{v}}{\|\boldsymbol{v}\|},$$
$$\text{Log}(\boldsymbol{q}) = 2\arccos(q_0)\frac{(q_x, q_y, q_z)}{\sqrt{q_x^2 + q_y^2 + q_z^2}},$$

where $\boldsymbol{v} \in \mathbb{R}^3$ is a vector representing an element in the Lie algebra $\mathfrak{su}(2)$, and $\boldsymbol{q}$ is a unit quaternion in $\mathcal{S}^3$.

Similar to rotation matrices, the exponential map for quaternions is surjective. Quaternions, which reside on the hypersphere $\mathcal{S}^3$, have special properties when restricted to a half-sphere. In this domain, the exponential and logarithmic maps are diffeomorphic, ensuring a one-to-one correspondence between the quaternion group and its Lie algebra.

6.2.2 NCDS on Lie Groups

Consider the situation where the system state represents its orientation, i.e. $\boldsymbol{x} \in \mathcal{SO}(3)$ or alternatively $\boldsymbol{x} \in \mathcal{S}^3$, whose first-order dynamics we seek to model with a latent NCDS. From a generative point of view, we first construct a decoder $\mu : \mathcal{Z} \to \mathfrak{so}(3)$ (or $\mu : \mathcal{Z} \to \mathfrak{su}(2)$ for quaternions) with outputs in the corresponding Lie algebra. We can then apply the exponential map to generate either a rotation matrix $\boldsymbol{R} \in \mathcal{SO}(3)$ or a quaternion $\boldsymbol{q} \in \mathcal{S}^3$, such that the complete decoder becomes $\mathrm{Exp} \circ \mu$ in both cases. Unfortunately, even if μ is injective, we cannot ensure that $\mathrm{Exp} \circ \mu$ is also injective (since Exp is surjective in both cases), which then breaks the stability guarantees of NCDS.

Here we leverage the result that Exp is a diffeomorphism as long as we restrict ourselves to the first cover of $\mathfrak{so}(3)$ or $\mathfrak{su}(2)$, depending on whether we are using rotation matrices or quaternions. Specifically, if we choose a decoder architecture such that $\mu : \mathcal{Z} \to \mathcal{B}_\pi$ is injective and has outputs on the first cover, then $\mathrm{Exp} \circ \mu$ is injective, and stability is ensured for both $\mathcal{SO}(3)$ and $\mathcal{S}^3$. To ensure that a decoder neural network has outputs over the π-balls, we introduce a simple layer. Let $\mu : \mathbb{R}^d \to \mathbb{R}^D$ be an injective neural network, where injectivity is only considered over the image of f. If we add a `TanH`-layer, then the output of the resulting network is the $[-1, 1]^D$ box, i.e. $\tanh(h(\boldsymbol{z})) : \mathbb{R}^d \to [-1, 1]^D$. We can further introduce the function,

$$b(\boldsymbol{x}) = \begin{cases} \frac{\|\boldsymbol{x}\|_\infty}{\|\boldsymbol{x}\|_2} \boldsymbol{x} & \boldsymbol{x} \neq \boldsymbol{0} \\ \boldsymbol{x} & \boldsymbol{x} = \boldsymbol{0} \end{cases}, \tag{6.4}$$

which smoothly (and invertibly) deforms the $[-1, 1]^D$ box into the unit ball (see Fig. 6.2–*right*). Then, the function,

$$h(\boldsymbol{z}) = \pi b(\tanh(\mu(\boldsymbol{z}))),$$

is injective and has the signature $h : \mathbb{R}^d \to \mathcal{B}_\pi(\boldsymbol{D})$.

During training, the observed rotation matrices can be mapped directly into the first cover of $\mathfrak{so}(3)$ using the logarithmic map, while quaternions can be mapped into the first cover of $\mathfrak{su}(2)$. When the system state consists of both orientations and positions, we

simply decode to higher-dimensional variables and apply exponential and logarithmic maps on the appropriate dimensions for both rotation matrices $\mathcal{SO}(3)$ and quaternions $\mathcal{S}^3$. Figure 6.1 illustrates the architecture using rotation matrices for convenience, but this is simply a design choice, as the orientation data in the preprocessing and postprocessing stages on the far left and far right of the architecture plot can easily be adapted to represent quaternions.

Algorithm 1: Task-Space Training Using $\mathcal{SO}(3)$

Input: Data: $\tau_n = \left\{ \boldsymbol{x}_{n,t}, \boldsymbol{R}_{n,t} \right\}_{n=1}^{N}, \quad t \in [1, T_n]$
Output: Learned contractive dynamics parameters $\boldsymbol{\theta}$.

foreach *trajectory n* **do**
  foreach *time step t* **do**
    $\boldsymbol{r}_{n,t} = \mathrm{Log}(\boldsymbol{R}_{n,t})$ ▷ Extract skew-sym coeffs
    $\boldsymbol{p}_{n,t} = [\boldsymbol{x}_{n,t}, \boldsymbol{r}_{n,t}]$ ▷ Form new state vector
  end
end
$\boldsymbol{\xi}^* = \arg\min_{\xi} \mathcal{L}_{\mathrm{ELBO}}(\boldsymbol{p}_{n,t})$ ▷ Train the VAE
foreach *trajectory n* **do**
  foreach *time step t* **do**
    $\boldsymbol{z}_{n,t} = \mu_{\xi^*}(\boldsymbol{p}_{n,t})$ ▷ Encode poses
    $\dot{\boldsymbol{z}}_{n,t} = \frac{\boldsymbol{z}_{n,t+1} - \boldsymbol{z}_{n,t}}{\Delta t}$ ▷ Compute latent velocities
  end
end
$\boldsymbol{\theta}^* = \arg\min_{\theta} \mathcal{L}_{\mathrm{Jac}}(\boldsymbol{p}_{n,t})$ ▷ Train Jacobian network

The steps for training the VAE and Jacobian network are outlined in Algorithm 1. Furthermore, the steps for employing NCDS to control a robot are detailed in Algorithm 2. As the algorithms show, the training of the latent NCDS is not fully end-to-end. In the latter case, we first train the VAE (end-to-end), and then train the latent NCDS using the encoded data.

Algorithm 2: Robot Control Scheme

Input: Data: $[\boldsymbol{x}_t, \boldsymbol{R}_t]$ ▷ Current state of the robot
Output: $\dot{\boldsymbol{x}}_t$ ▷ Velocity of the end-effector
foreach *time step* t **do**
  $\boldsymbol{r}_t = \text{Log}(\boldsymbol{R}_t)$ ▷ Extract skew-sym coeffs
  $\boldsymbol{p}_t = [\boldsymbol{x}_t, \boldsymbol{r}_t]$ ▷ Form new state vector
  $\boldsymbol{z}_t = \mu_\xi(\boldsymbol{p}_t)$ ▷ Compute the latent state
  $\hat{\dot{\boldsymbol{z}}}_t = f(\boldsymbol{z}_t)$ ▷ Compute the latent velocity
  $\boldsymbol{J}_{\mu_\xi}(\boldsymbol{z}_t) = \frac{\partial \mu_\xi}{\partial \boldsymbol{z}_t}$ ▷ Compute the Jacobian of the decoder
  $\dot{\boldsymbol{x}}_t = \boldsymbol{J}_{\mu_\xi}(\boldsymbol{z}_t)\hat{\dot{\boldsymbol{z}}}_t$ ▷ Compute input space velocity
end

6.3 Riemannian Safety Regions and Latent Obstacle Avoidance

Section 2.3 described a Riemannian metric (2.13) in the VAE latent space. Under this metric, shortest paths stay within the data support while avoiding obstacles. Building on the approach for learning Riemannian manifolds from data, we now introduce a *modulation matrix* that ensures our latent NCDS vector field also stays within the data support while avoiding obstacles. Similar to classical robotics, where the configuration space provides a representation distinct from the task space for obstacle avoidance, our approach constructs a low-dimensional representation of the task space on which we perform obstacle avoidance by pulling back information from the ambient (robot's end-effector) task space. We notice that the volume of the pullback Riemannian metric in (2.13) increases when moving away from the data manifold and when approaching an obstacle. If we change the NCDS vector field to avoid "high volume" areas, our goal will be achieved. We accomplish this via a Riemannian modulation matrix $\boldsymbol{G}_{\mathcal{M}}(\boldsymbol{z})$ that reshapes the vector field f, resulting in an obstacle-free vector field $\hat{f}$,

$$\hat{f}(\boldsymbol{z}) = \boldsymbol{G}_{\mathcal{M}}(\boldsymbol{z}) f(\boldsymbol{z}). \tag{6.5}$$

To design this modulation matrix, we follow the recipe of Huber et al. (2022), described in Sec. 2.7, and let $\boldsymbol{G}_{\mathcal{M}}(\boldsymbol{z}) = \boldsymbol{E}(\boldsymbol{z})\boldsymbol{D}(\boldsymbol{z})\boldsymbol{E}(\boldsymbol{z})^{-1}$.

First, the matrix $\boldsymbol{D}$ is designed to guarantee *impenetrability* and to ensure the *local effect* of the modulation matrix. We use the formulation of Huber et al. (2022), which is given in (5.12). Specifically, we ensure that $\lambda_n(\boldsymbol{z}) : \mathbb{R}^d \to [0.0, 1.0]$ decreases towards

0.0 and $\lambda_\tau(\boldsymbol{z}) : \mathbb{R}^d \rightarrow [1.0, 2.0]$ increases towards 2.0 when the distance to the obstacle decreases. Based on these requirements, matrix $\boldsymbol{D}$ can be designed according to the following sigmoid function,

$$\Xi(\rho, \nu, \lambda_{init}, \lambda_{end}, k) = \lambda_{init} + \tag{6.6}$$

$$\frac{\lambda_{end} - \lambda_{init}}{1 + \exp\left(-k\left[S(\boldsymbol{x}) - \frac{(\rho+\nu)}{2}\right]\right)}, \text{with} \tag{6.7}$$

$$\lambda_n(\boldsymbol{x}) = \Xi(\rho = 1, \nu = 10, \lambda_{init} = 0, \lambda_{end} = 1, k = 2),$$

$$\lambda_\tau(\boldsymbol{x}) = \Xi(\rho = 1, \nu = 10, \lambda_{init} = 2, \lambda_{end} = 1, k = 2).$$

The parameters are defined as follows: ρ specifies the distance at which the obstacle becomes impenetrable, while ν indicates the distance from which the modulation is inactive. The initial and final values of the sigmoid function $\Xi(...)$ are given by λ_{init} and λ_{end}, respectively. For example, in Fig. 6.3, λ_{init} and λ_{end} (shown in blue) are the initial and target values for λ_τ. This means that λ_τ takes the value of λ_{init} when inside the obstacle and similarly it takes the values of λ_{end} when outside of the obstacle. The same applies for λ_n, depicted in orange. Lastly, k controls the smoothness of the transition between these values. The specific values for these parameters are provided in Koptev (2023) to configure the matrix $\boldsymbol{D}$ for correct modulation activation. In this specific setup, the modulation activates ν units away from the obstacle surface and progressively intensifies until it reaches ρ unit from the surface, at which point the surface becomes impenetrable. These values can be adjusted to suit different configurations or experimental conditions. Figure 6.3 illustrates how the elements of matrix $\boldsymbol{D}$ behave as a function of the distance from the obstacle. Note that the values of $\lambda_n(\boldsymbol{x})$ begin to decrease towards 0.0, while $\lambda_t(\boldsymbol{x})$ values start increasing towards 2.0 as the distance drops below ν units.

Second, the basis matrix $\boldsymbol{E}$, which determines the modulation directions, is defined by stacking the obstacle normal $\boldsymbol{n}$ (i.e. a vector orthogonal to the tangent plane of the obstacle surface, pointing outward) and an orthogonal basis vectors $\boldsymbol{e}$ that defines a hyperplane tangential to the surface of the obstacle. To allow for obstacles with complex shapes, we compute the normal through a distance field $\mathfrak{S}$ that determines the distance from any point to the obstacle (Koptev, 2023). The normal vector can then be chosen as,

$$\boldsymbol{n}(\boldsymbol{z}) = \nabla\mathfrak{S}(\boldsymbol{z}) \quad \text{s.t.} \quad \boldsymbol{e}(\boldsymbol{z}) \perp \boldsymbol{n}(\boldsymbol{z}), \tag{6.8}$$

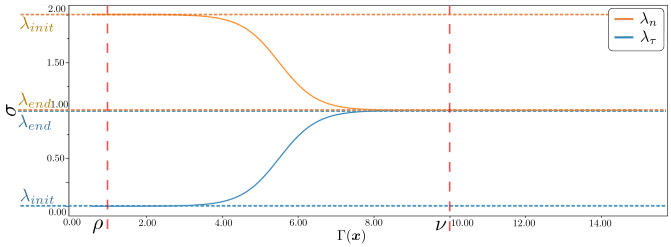


Figure 6.3: The behavior of the elements of the matrix $\boldsymbol{D}$ in response to distance from obstacle.

where $\nabla\mathfrak{S}(\boldsymbol{z})$ is the gradient of the distance field at $\boldsymbol{z}$. To incorporate the Riemannian metric, we rescale the distance field by the inverse Riemannian volume,

$$\mathfrak{S}_{\text{scaled}}(\boldsymbol{z}) = \frac{\alpha}{\mathcal{V}(\boldsymbol{z})}\mathfrak{S}(\boldsymbol{z}), \quad \text{with} \quad \mathcal{V}(\boldsymbol{z}) = \sqrt{|\det(\boldsymbol{M}(\boldsymbol{z}))|},$$

where α is a scaling factor.

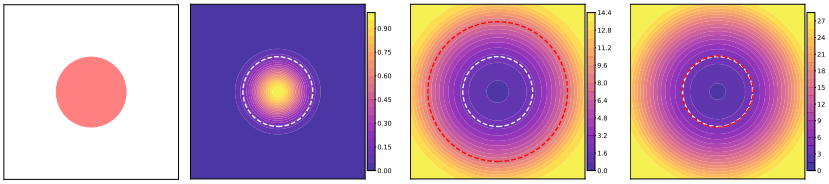


Figure 6.4: Illustration of the computation of the obstacle avoidance distance field for designing Riemannian safety regions. From left: (1) Obstacle representation (circle), (2) Metric volume around obstacle, (3) Metric inversion to compute distance map and problem with initial boundary misalignment (white vs. red circle for real vs. approximated boundary), (4) Rescaling with α to correct the boundary.

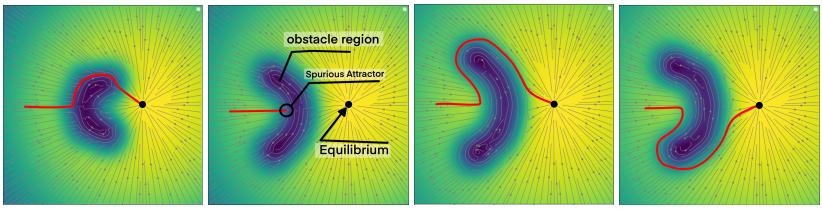
Figure 6.4 illustrates the computation of the distance field. To compute the weight α, we discretize the data manifold with an equidistant mesh grid in the latent space $\mathcal{Z}$, followed by the computation of the corresponding metric volumes $\mathcal{V}(\boldsymbol{z})$ for each point on the mesh grid. Later, the maximum and the minimum volumes, $\mathcal{V}_{\text{max}}$ and $\mathcal{V}_{\text{min}}$, are used to normalize all volumes $\mathcal{V}(\boldsymbol{z})$ as follows,

$$\mathcal{V}(\boldsymbol{z}) = \frac{\mathcal{V}(\boldsymbol{z}) - \mathcal{V}_{\text{min}}}{\mathcal{V}_{\text{max}} - \mathcal{V}_{\text{min}}}.$$

Afterward, α is experimentally chosen to align with the specified distance range defined by ρ and ν in (6.7). This ensures that the scalar field is scaled appropriately, such that:

$$\mathfrak{S}(\boldsymbol{z}) < \rho, \quad \boldsymbol{z} \in \text{obstacle region},$$
$$\mathfrak{S}(\boldsymbol{z}) > \nu, \quad \boldsymbol{z} \notin \text{obstacle region}.$$

The complete process from obstacle to distance field involves several key steps, as illustrated in Fig. 6.4. The obstacle is presented here as a circle for simplicity (leftmost panel), but it can be represented using more complex shapes, such as meshes, depending on the application. A Gaussian-like function is then applied to reshape the manifold, serving as the ambient metric centered on the obstacle (second panel). This metric models the obstacle's influence on the surrounding space. The metric is inverted to construct a distance map, as shown in the third panel. However, the computed obstacle boundary (red circle) does not align with the real boundary (white circle) due to the nature of the Gaussian function, which skews distance measurements. Finally, in the fourth panel, the distance field is rescaled using the parameter α. This parameter adjusts the Gaussian's influence, ensuring the computed boundary aligns with the real obstacle boundary, regardless of the obstacle's shape.



Convex obstacle without tangential vector field | Concave obstacle without tangential vector field | Concave obstacle with left tangential vector field | Concave obstacle with right tangential vector field

Figure 6.5: Modulated contractive dynamical systems using a pullback metric derived from the decoder's Jacobian.

As Koptev (2023) discuss, the modulation in (5.10) can generate spurious attractors around concave regions of the obstacle's surface (Fig.6.5–*b*). Koptev (2023) suggests using a secondary tangential vector field that activates only when the velocity is zero

to avoid this issue. Specifically,

$$\hat{f} = \boldsymbol{G}_{\mathcal{M}}(\boldsymbol{z}) \cdot f(\boldsymbol{z}) + \beta(\boldsymbol{x})\boldsymbol{G}_{\mathcal{M}}(\boldsymbol{z}) \cdot g(\boldsymbol{z}), \quad \beta \in [0,1] \tag{6.9}$$

where β increases as the velocity generated by the main modulation approaches zero. This tangential vector field is computed using the tangent vector on the surface of the obstacle. When multiple tangent vector fields are available, it is crucial to select the one that produces the most optimal modulated vector field. For example, in a two-dimensional space (as illustrated in Fig. 6.5–*c* and *d*), two distinct tangent vector fields can be identified. The criteria for choosing the appropriate tangent vector field are highly problem-dependent. For instance, when the criterion is to follow the shortest path to the target, one can compute the geodesic path and then select the tangential direction that most closely aligns with it. Next, we will evaluate the performance of latent NCDS across various experimental settings.

6.4 Experiments

To evaluate the efficiency of latent NCDS, we consider several synthetic and real tasks. Comparatively, we show that NCDS is the only method to scale gracefully to higher-dimensional problems, due to the latent structure. Through ablation studies in Sec. 5.6.2, we further analyze the effect of various activation functions, regularization techniques, and network architectures on the NCDS performance. We further demonstrate the ability to build dynamic systems on the Lie group of rotations and to avoid obstacles while ensuring stability. None of the baseline methods has such capabilities.

Datasets.

In Section 5.6, we conducted preliminary experiments on the 2D LASA dataset. To further complicate this easy task, we aim at learning NCDS on 2, or 4 stacked trajectories, resulting in 4 (LASA-4D), or 8-dimensional (LASA-8D) data, respectively. Secondly, we consider a dataset consisting of 5 trajectories collected via kinesthetic teaching on a 7-DoF Franka-Emika Panda robot. These trajectories form a V-shape path on the desk surface, with the orientation of the end-effector smoothly changing to follow the direction of motion. To evaluate the model's performance in a more challenging setting, we use the KIT Whole-Body Human Motion Database (Mandery et al., 2016). We provide further details on how these datasets are used in the relevant sections.

Metrics

We use dynamic time warping distance (DTWD) as the established quantitative measure of reproduction accuracy w.r.t. a demonstrated trajectory, assuming equal initial conditions (Ravichandar et al., 2017, Sindhwani et al., 2018),

$$\begin{aligned}\mathrm{DTWD}(\tau_x, \tau_{x'}) = &\sum_{j \in l(\tau_{x'})} \min_{i \in l(\tau_x)} \left(d(\tau_{x_i}, \tau_{x'_j}) \right) + \\ &\sum_{i \in l(\tau_x)} \min_{j \in l(\tau_{x'})} \left(d(\tau_{x_i}, \tau_{x'_j}) \right),\end{aligned}$$

where τ_x and $\tau_{x'}$ are two trajectories (e.g. a path integral and a demonstration trajectory), d is a distance function (e.g. Euclidean distance), and $l(\tau)$ is the length of trajectory τ. To quantitatively assess the contraction properties of different vector fields, we introduce a metric based on integral curves. Specifically, we consider N integral curves, each initiated from points on an equidistant grid in the state space. We rollout the trajectory over a period of T time steps. We define a *demonstration region* by computing the convex hull $\mathcal{H}$ around a set of demonstration data. In practice, we utilize the computational geometry libraries such as `shapely`[1] to handle these operations. The convex hull $\mathcal{H}$ is computed as a polygon using `shapely.geometry.Polygon`. Point membership is then efficiently determined using standard point-in-polygon algorithms (e.g., ray-casting). The hyperparameters determining this convex hull shape, such as the margin size extending beyond the data, are chosen experimentally to best capture the region of interest. For each integral curve $\mathbf{x}_i(t)$, where $i = 1, \dots, N$ and $t = 1, \dots, T$, we compute the number of time steps n_i for which the trajectory resides within $\mathcal{H}$:

$$n_i = \sum_{t=1}^{T} \mathfrak{b}\Big(\mathbf{x}_i(t) \in \mathcal{H} \Big), \tag{6.10}$$

where $\mathfrak{b}(\cdot)$ is the indicator function, returning 1 if its argument is true and 0 otherwise. Intuitively, the more time steps a trajectory spends within $\mathcal{H}$, the faster it converges toward it, indicating a higher contractive behavior towards the demonstrations region. Additionally, we introduce quantitative metrics to directly evaluate the system's contraction properties. Specifically, we compute the contraction ratio as the maximum

[1] Shapely: Python package. Last checked: February 2025.

absolute difference between the eigenvalues $\Delta\lambda = \lambda_{\max} - \lambda_{\min}$. Note that the the contraction ratio and rate reflect the overall contraction properties of the system, rather than its convergence toward the data support region.

Baseline methods

Our work is the first contractive neural network architecture, so we cannot compare NCDS directly to methods with identical goals. Indeed, we report in Table 6.1 the main papers in the contraction literature, showing that a lack of consensus exists regarding the method or baseline for comparison, in the context of learning contractive dynamical systems. However, we compare our method to *ELCD*, recently introduced by Jaffe et

Table 6.1: The present state-of-the-art literature pertaining to the learning of contractive dynamical systems.

Paper Title	Abbr.	Compared Against	Year
Safe Control with Learned Certificates: A Survey of Neural Lyapunov, Barrier, and Contraction Methods (Dawson et al., 2023)			2022
Learning Contraction Policies from Offline Data (Rezazadeh et al., 2022)		MPC (ILQR), RL (CQL)	2022
Neural Contraction Metrics for Robust Estimation and Control: A Convex Optimization Approach (Tsukamoto et al., 2021b)	NCM	CV-STEM, LQR	2021
Learning Stabilizable Nonlinear Dynamics with Contraction-Based Regularization (Singh et al., 2021)	CCM-R		2021
Learning Certified Control Using Contraction Metric (Sun et al., 2020)	C3M	SoS (Sum-of-Squares programming), MPC, RL (PPO)	2020
Learning Position and Orientation Dynamics from Demonstrations via Contraction Analysis (Ravichandar et al., 2019)	CDSP	SEDS, CLF-DM, Tau-SEDS, NIVF	2019
Learning Stable Dynamical Systems Using Contraction Theory (Blocher et al., 2017)	C-GMR	SEDS	2017
Learning Contracting Vector Fields for Stable Imitation Learning (Sindhwani et al., 2018)	CVF	DMP, SEDS, CLF-DM	2017
Learning Partially Contracting Dynamical Systems from Demonstrations (Ravichandar et al., 2017)	CDSP (position only)	DMP, CLF-DM	2017

al., 2024, which leverages normalizing flows for learning contraction. Instead, we compare to existing methods that provide asymptotic stability guarantees. In particular, *Euclideanizing flow* (Rana et al., 2020b), *Imitation flow* (Urain et al., 2020) and *SEDS* (Khansari-Zadeh et al., 2011a). For Imitation flow, we use a network of 5 layers, where each was constructed using CouplingLayer, RandomPermutation, and LULinear techniques, as recommended by (Urain et al., 2020) for managing high-dimensional data. We train for 1000 epochs, with a learning rate of 10^{-3}. For Euclidenizing flow, we used a coupling network with random Fourier features, using 10 coupling layers of 200 hidden units, and a length scale of 0.45 for the random Fourier features. This is trained for 1000 epochs, with a learning rate of 10^{-4}. Lastly, for SEDS, we use a Gaussian mixture model of 5 components that are trained for 1000 epochs with an MSE objective. In all cases, hyperparameters were found experimentally and the best results have been selected to be compared against. For ELCD, we used the public codebase[1]. We followed their suggested parameter set, using 2 flow steps and a hidden dimension of 16. The model was trained for 100 epochs with a learning rate of 10^{-3}.

Table 6.2: Average dynamic time warping distances (DTWD) between different approaches. NCDS shows competitive performance in low-dimensional spaces and outperforms others in higher dimensions.

Dataset	LASA-2D	LASA-4D	LASA-8D	7 DoF Robot
Euclideanizing Flow	$\mathbf{0.72 \pm 0.12}$	3.23 ± 0.34	10.22 ± 0.40	5.11 ± 0.30
Imitation Flow	0.80 ± 0.24	$\mathbf{0.79 \pm 0.22}$	4.69 ± 0.52	2.63 ± 0.37
SEDS	1.60 ± 0.44	3.08 ± 0.20	4.85 ± 1.64	2.69 ± 0.18
NCDS	1.37 ± 0.40	0.98 ± 0.15	$\mathbf{2.28 \pm 0.24}$	$\mathbf{1.18 \pm 0.16}$

Table 6.3: Average execution time (in milliseconds) of a single NCDS integration step for learning the vector field in different spaces.

Input Dim	2D	3D	8D	44D
Input Space (Without VAE)	1.23 ms	-	40.9 ms	-
Latent Space (With VAE)	-	10.6 ms	20.2 ms	121.0 ms

[1] ELCD codebase: GitHub Repository. Last checked: February 2025.

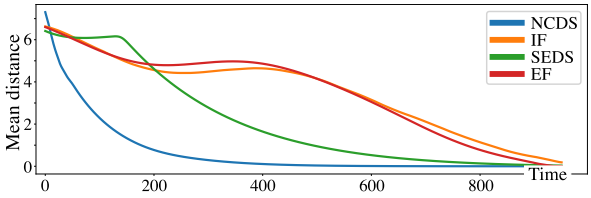


Figure 6.6: Average distance between random nearby trajectories over time for LASA-2D. Only NCDS monotonically decreases, i.e. it is the only contractive method.

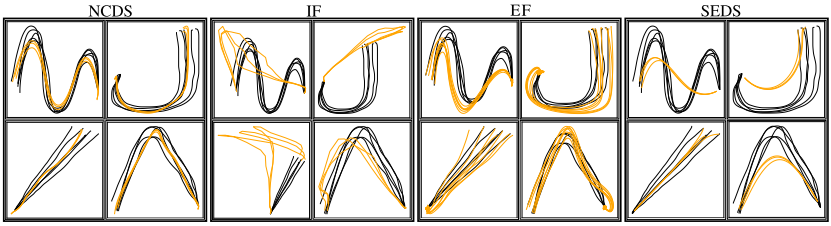


Figure 6.7: Path integrals generated using different methods on LASA-8D. Black curves are training data, while yellow are the learned path integrals. To construct the 8D dataset, we have concatenated 4 different 2D datasets.

6.4.1 Comparative Study on the LASA Dataset

Table 6.2 shows average DTWD distances among five generated path integrals and five demonstration trajectories for both NCDS and baseline methods. For the two-dimensional problem, both Euclideanizing flow and Imitation flow outperform NCDS and SEDS, though all methods perform quite well. To investigate stability, Fig. 6.6 displays the average distance over time among five path integrals starting from random nearby initial points for different methods. We observe that only for NCDS does this distance decrease monotonically, which indicates that it is the only contractive method. To test how these approaches scale to higher-dimensional settings, we consider the LASA-4D and LASA-8D datasets, where we train NCDS with a two-dimensional latent representation. From Table 6.2, it is evident that the baseline methods quickly deteriorate as the data dimension increases, and only NCDS gracefully scales to higher dimensional data. This is also evident from Fig. 6.7, which shows the training data and reconstructed trajectories of different LASA-8D dimensions with different methods. These results clearly show the value of having a low-dimensional latent structure.

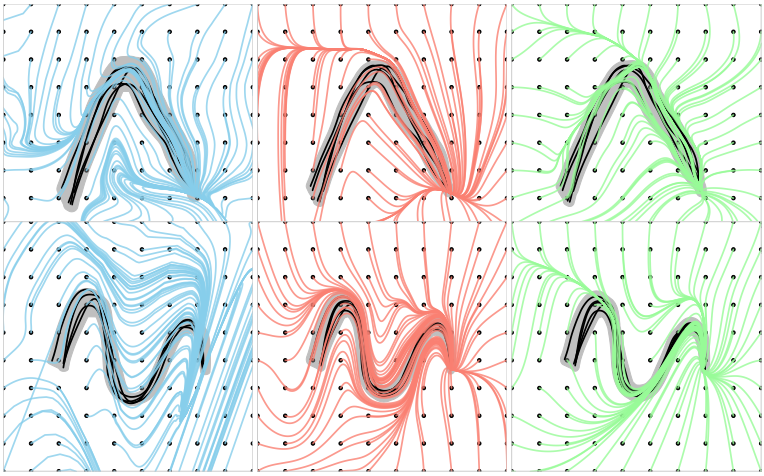
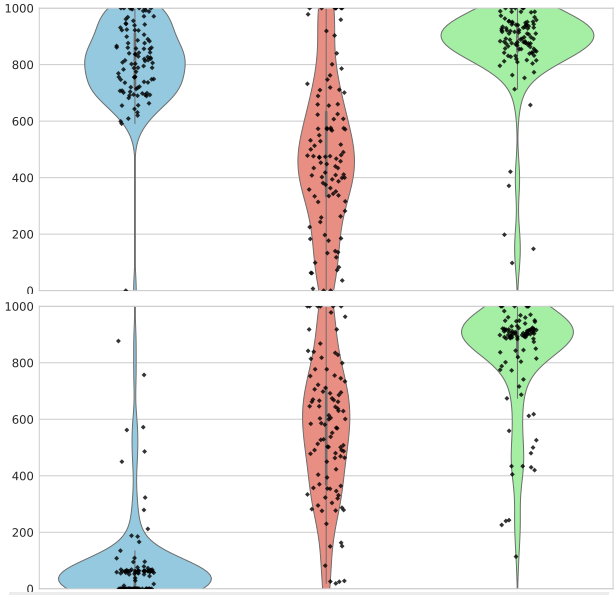


Figure 6.8: Comparison of vector fields generated by NCDS and ELCD (Jaffe et al., 2024). The left, middle, and right columns depict the integral curves produced by ELCD, vanilla NCDS, and NCDS with state-independent regularization, respectively. A total of 100 integral curves, each consisting of 1000 points, are generated on an equidistant grid around the demonstration data and evaluated on two LASA datasets: *Angle* and *Sine*.

Next, we compare our method to ELCD (Jaffe et al., 2024) using two different LASA datasets. Figure 6.8 illustrates 100 integral curves, each generated on an equidistant grid with 1000 time steps. The results indicate that all methods successfully recover the demonstration while maintaining contractivity. However, ELCD shows unnecessary deviations outside the data support. Instead of smoothly contracting toward the demonstrations region, some trajectories take detours, making extra turns or oscillations before eventually converging.

Figure 6.9-*top* quantifies how many of the 1000 points in each integral curve remain inside the demonstration region. A higher value means that the trajectory stays within the data support for more time steps, indicating stronger contraction. The table in Fig. 6.9-*bottom* presents the average number of points spent within the demonstration region across all integral curves.

The results show that NCDS with state-independent regularization keeps trajectories more focused on the demonstrations region. Particularly when using the *Angle* dataset, ELCD exhibits stronger contraction than vanilla NCDS, as shown in Fig. 6.9-*bottom*. However, as illustrated in Fig. 6.8, ELCD also introduces more unnecessary motion



Method	Angle	Sine
ELCD	809	82
Vanilla NCDS	484	561
Independent Reg.	**862**	**824**

Figure 6.9: Comparison of contractivity across methods. The top and middle plots show the number of trajectory points inside the demonstration region for models trained on *Angle* and *Sine* datasets, respectively, where higher values indicate stronger contraction. The bottom table presents the average number of points per integral curve within the demonstration region.

patterns outside the demonstration region. Moreover, Fig. 6.9-*middle* shows the results for the *Sine* dataset, where ELCD, despite our efforts to enhance its performance, does not perform as well. This trend is further supported by Fig. 6.9-*bottom*.

Comparative study on the robot dataset

To further compare our method, we consider a robotic setting where we evaluate all the methods on a real dataset of joint-space trajectories collected on a 7-DoF robot. The last column of Table 6.2 shows that only NCDS scales gracefully to high-dimensional

joint-space data, and outperforms the baseline methods. The supplementary material includes a video showcasing the simulation environment in which the robot executes a drawing task, depicting V-shape trajectories on the table surface.

Dimensionality and execution time

Table 6.3 compares 5 different experimental setups. Both the Variational Autoencoder (VAE) and Jacobian network $\hat{\boldsymbol{J}}_\theta$, were implemented using PyTorch (Paszke et al., 2019b). The VAE employs an injective generator based on $\mathcal{M}$-flows (Brehmer et al., 2020), using rational-quadratic neural spline flows with three coupling layers. The experiments were conducted on a system with an Intel® Xeon® Processor E3-1505M v6 (8M Cache, 3.00 GHz), 32 GB of RAM, and an NVIDIA Quadro M2200 GPU.

Table 6.3 reports the average execution time when dimensionality reduction is not used and the dynamical system is learned directly in the input space. As the results show, increasing the dimensionality of the input space from 2 to 8 entails a remarkable 40-fold increase in execution time for a single integration step. This empirical relationship reaffirms the inherent computational intensity tied to Jacobian-based computations. This table also shows that our VAE distinctly mitigates the computational burden. In comparison, the considerable advantages of integrating the contractive dynamical system with the VAE into the full pipeline become evident as it leads to a significant halving of the execution time in the 8D scenario. We further find that increasing the dimensionality from 8 to 44 results in a sixfold increase in running time. The results indicate that our current system may not achieve real-time operation. However, the supplementary video from our real-world robot experiments demonstrates that the system's rapid query response time is sufficiently fast to control the robot arm, devoid of any operational concerns.

6.4.2 Generalization Outside the Decoder's Manifold

In our framework, although the latent and ambient spaces share the same overall dimensionality due to the padding operation, the encoder projects data into a lower-dimensional representation by discarding certain dimensions (via unpadding). These discarded (or extra) dimensions, while not directly used in the final projection, retain information that is indicative of whether a point lies on the decoder's manifold. We define the decoder's manifold as the set of latent points for which the extra dimensions are exactly zero. Formally, let $\boldsymbol{z} = [\boldsymbol{z}_m, \boldsymbol{z}_e]$ be the latent vector, where $\boldsymbol{z}_m$ represents

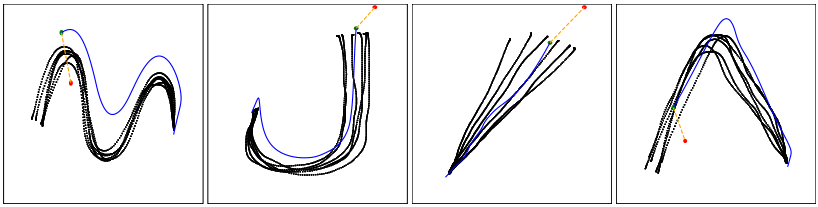


Figure 6.10: Generalization under deviation from the decoder's manifold using the 8D LASA dataset. Red circles indicate initial positions, the orange dashed line shows the transition vector to the manifold (green circle), and the blue curve represents the integral curve.

the dimensions used for the final projection and $\boldsymbol{z}_e$ the extra dimensions, then a point is considered on the decoder's manifold if $\| \boldsymbol{z}_e \| = 0$.

Conversely, any nonzero values in $\boldsymbol{z}_e$ indicate that the corresponding state lies outside the decoder's manifold. Although deviations from the decoder's manifold are generally minimal, in certain real-robot scenarios measurement noise or ambient perturbations can occasionally cause the latent representation to stray from the decoder's manifold. In these cases, our approach incorporates a transition phase to correct the deviation. Specifically, if a latent point is identified as being off the manifold, we compute a transition vector that incrementally steers the point back toward the manifold. This is achieved by integrating along a path in latent space until the condition $\| \boldsymbol{z}_e \| \simeq 0$ is effectively reached, thereby ensuring compatibility with the decoder's assumptions. This corrective mechanism is not invoked routinely but serves as an important safeguard to enhance system robustness in the rare instances when deviations occur.

Figure 6.10 illustrates this procedure using the 8D LASA dataset. In the figure, red markers denote the initial latent positions that deviate from the demonstration's starting point. The orange dashed line represents the transition vector that guides the point toward the manifold, and the green marker indicates the latent point after successful projection onto the manifold. The blue curve depicts the integrated path taken during the transition phase.

By quantifying the deviation from the manifold e.g., via the Euclidean norm $\| \boldsymbol{z}_e \|$, we can effectively measure and correct the discrepancy between the actual latent representation and the ideal manifold state.

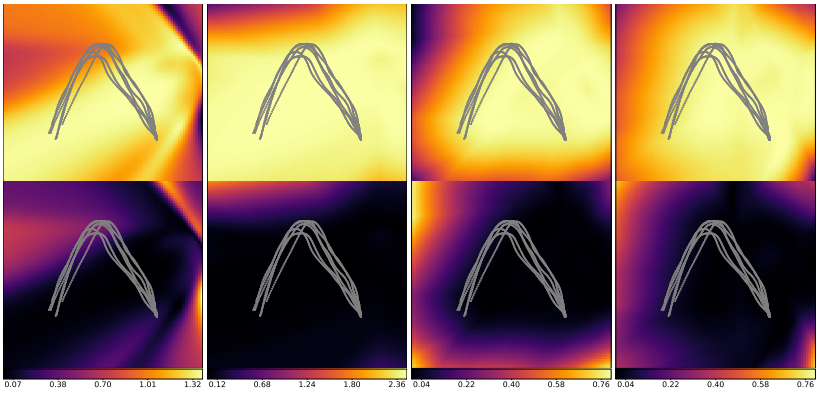


Figure 6.11: Comparison of 2D vector fields learned by using the following regularization approaches: **(a)** constant regularization, **(b)** state-independent regularization (using basic optimization), **(c)** state-dependent regularization (using a neural network), and **(d)** eigenvalue regularization. The *top row* displays a map of the maximum eigenvalue, while the *bottom row* shows the absolute difference between eigenvalues (i.e., the contraction ratio).

6.4.3 Eigenvalue Metric Maps of Regularization Methods

To better assess the impact of each regularization method, Fig. 6.11 presents heatmaps depicting the distribution of various eigenvalue metrics within a selected square region around the demonstration data. The results for state-independent regularization show that this method exhibits larger negative values across the spectrum, indicating overall stronger contraction. Specifically, within the selected region, the contraction rate reaches -0.15, the highest among all methods. Additionally, as shown in the figure, this method achieves the largest contraction ratio, suggesting a more directionally dependent contractive behavior.

6.4.4 Eigenvalue Metric Maps of Symmetric vs. Asymmetric Jacobian

Figure 6.12 shows different eigenvalue metric heatmaps of the learned contractive dynamics. The top row shows that symmetric Jacobian approach results in increased contraction along the “dominant” eigenvector. In the middle row, we observe that even the “non-dominant” eigenvector has stronger contraction under this method.

Finally, the bottom row shows a higher overall contraction ratio, indicating a more pronounced pull toward a specific eigenvector.

More precisely, in the *Angle* dataset, the average minimum eigenvalue is -0.266 compared to -0.046 for the asymmetric case, while the average maximum eigenvalue remains more negative (-0.034 vs. -0.006). The maximum contraction ratio, representing the difference between these eigenvalues, is significantly higher for the symmetric model (2.489 vs. 0.362), confirming its more robust contractive behavior. Extreme values further support this trend, with the minimum eigenvalue reaching -2.589 for the symmetric case, compared to only -0.388 for the asymmetric model.

Similarly, in the *Sine* dataset, the symmetric model maintains a average minimum eigenvalue of -0.381 versus -0.063 in the asymmetric case, with a more negative average maximum eigenvalue (-0.042 vs. -0.003). The maximum contraction ratio is also notably higher (1.457 vs. 0.439), reinforcing the role of symmetry in promoting directional contraction. The extreme minimum eigenvalue reaches -1.489 in the symmetric formulation, whereas it remains at -0.439 for the asymmetric model.

6.4.5 Learning Robot Motion Skills

To demonstrate the application of NCDS to robotics, we conducted several experiments on a 7-DoF Franka-Emika robotic manipulator, where an operator kinesthetically teaches the robot drawing tasks. Both the Variational Autoencoder (VAE) and the Jacobian network $\hat{\boldsymbol{J}}_{\boldsymbol{\theta}}$, were implemented using the PyTorch framework (Paszke et al., 2019b). For the VAE, we used an injective generator based on $\mathcal{M}$-flows (Brehmer et al., 2020), specifically employing rational-quadratic neural spline flows with three coupling layers. In each coupling transform, half of the input values underwent element-wise transformation through a monotonic rational-quadratic spline. These splines were parameterized by a residual network with two residual blocks, each containing a hidden layer of 30 nodes. We used `Tanh` activations throughout the network, without batch normalization or dropout. The rational-quadratic splines used ten bins, evenly spaced over the range $(-10, 10)$. To learn these skills, we employ the same architecture described in Sec. 5.6.1. The learned dynamics are executed using a Cartesian impedance controller. First, we demonstrate 7 sinusoidal trajectories while keeping the end-effector orientation constant, such that the robot end-effector dynamics only evolve in $\mathbb{R}^3$.

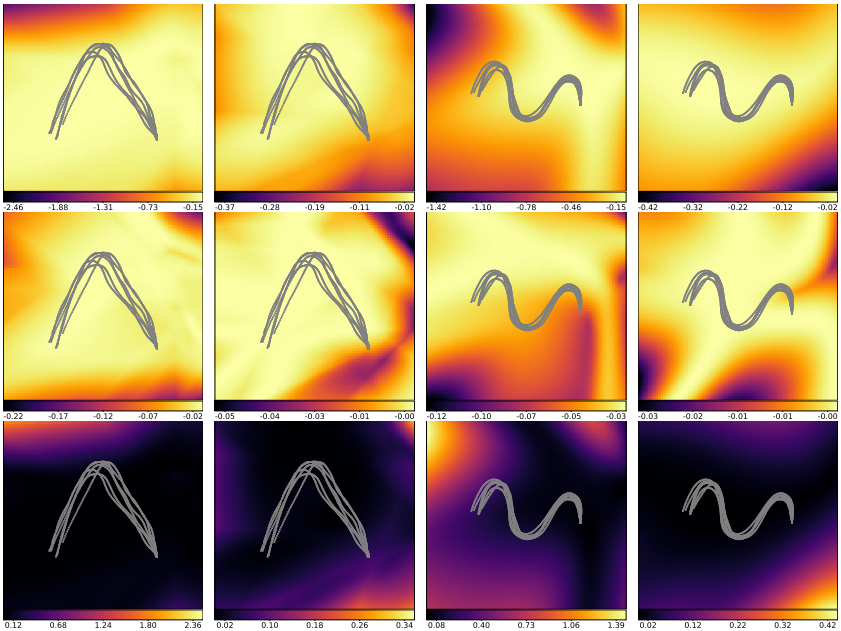


Figure 6.12: Eigenvalue metric heatmaps of the learned contractive dynamics shown in Fig. 5.6. The *top* and *middle* rows display the minimum and maximum eigenvalues computed at each point on the equidistant grid, respectively. The *bottom* row shows the contraction ratio, defined as the absolute difference between these eigenvalues. The first and third columns correspond to the results obtained with a symmetric Jacobian for the *Angle* and *Sine* datasets, respectively, while second and forth columns show the corresponding results for an asymmetric Jacobian.

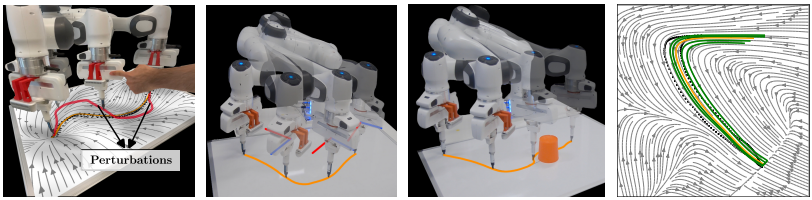


Figure 6.13: Robot experiments. The left two panels show robot experiments with orange and red paths illustrating unperturbed and perturbed motion. The first panel shows a learned vector field in $\mathbb{R}^3$ with a constant orientation. In the second panel, the vector field expands to $\mathbb{R}^3 \times \mathcal{SO}(3)$, aligning the end-effector's orientation with the motion direction. Superimposed robot images depict frames of executed motion. The third panel shows the robot successfully avoiding the obstacle (orange cylinder) using the modulation technique utilized in the Vanilla NCDS. The right panel illustrates the contours of the latent vector field in the background, with the green and yellow curves representing the path integrals, while the black dots indicate the demonstrations in latent space.

As Fig. 6.13-*left* shows, the robot was able to successfully reproduce the demonstrated dynamics by following the learned vector field encoded by NCDS. Importantly, we also tested the extrapolation capabilities of our approach by introducing physical perturbations to the robot end-effector, under which the robot satisfactorily adapted and completed the task (see the supplementary video).

The second experiment tests NCDS ability to learn orientation dynamics evolving in $\mathbb{R}^3 \times \mathcal{SO}(3)$, i.e. full-pose end-effector movements. We collect several V-shape trajectories on a table surface on which the robot end-effector always faces the direction of the movement, as shown in the second panel from the left in Fig. 6.13. However, the orientation trajectories of the robot experiments tend to show relatively simple dynamics. To evaluate the performance of this setup on a more complex dataset, we generate a synthetic LASA dataset in $\mathbb{R}^3 \times \mathcal{SO}(3)$. To introduce complexity within $\mathcal{SO}(3)$, we project the LASA datasets onto a 3-sphere, thus generating synthetic quaternion data. Although we consider orientation in $\mathcal{SO}(3)$ here, it is worth noting that NCDS can also handle quaternions directly. Next, we transform these into rotation matrices on $\mathcal{SO}(3)$, and apply the Log map to project onto the Lie algebra. To construct the position data in $\mathbb{R}^3$, we embed the 2D LASA dataset in $\mathbb{R}^3$ by concatenating with a zero. Together, this produces a state vector $\boldsymbol{x}$ in $\mathbb{R}^6$. To increase the level of complexity, we employ two distinct LASA letter shapes for the position and orientation dimensions.

Figure 6.13-*right*, depicts the 2D latent dynamical system, where the black dots represent the demonstrations. Moreover, the yellow path integrals start at the initial point of the demonstrations, while the green path integrals start at random points around the data support. This shows that NCDS can learn complex nonlinear contractive dynamical systems in $\mathbb{R}^3 \times \mathcal{SO}(3)$.

6.4.6 Obstacle Avoidance in Vanilla NCDS

Unlike baselines methods, our Vanilla NCDS framework also provides dynamic collision-free motions. To demonstrate this in practice, we experiment with a real robot, where we block motion demonstrations with an object. Here, the modulation matrix is computed based on the formulation in (5.10). The third panel of Fig. 6.13 shows the obstacle avoidance in action as the robot's end-effector navigates around the orange cylinder. This demonstrates how the dynamic modulation matrix approach leads to obstacle-free trajectories while guaranteeing contraction.

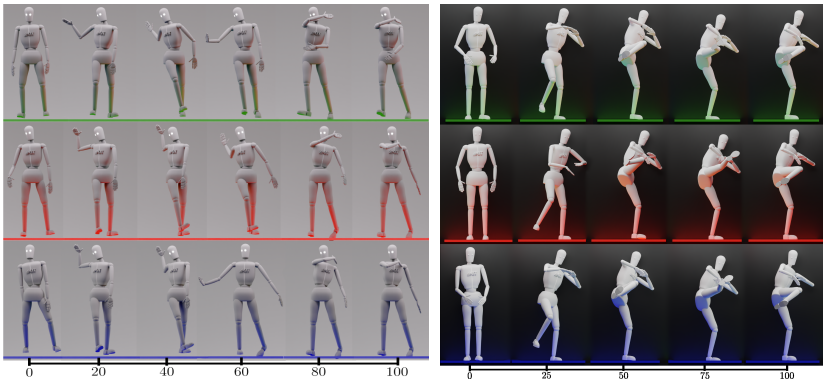


Figure 6.14: *Left*: Generated Human Motion with NCDS trained in joint space $\mathbb{R}^{44}$: The upper row depicts the progressive evolution of the demonstration motion from left to right. Meanwhile, the middle and bottom rows illustrate the motions generated by NCDS when the initial point is respectively distant from and coincident with the initial point of the demonstration. *Right*: Generated Human Motion with NCDS trained on full human motion space $\mathbb{R}^{44} \times \mathbb{R}^3 \times \mathcal{SO}(3)$: The upper row depicts the progressive evolution of the demonstration motion from left to right. Meanwhile, the middle and bottom rows illustrate the motions generated by NCDS when the initial point is respectively distant from and coincident with the initial point of the demonstration.

6.4.7 Learning Human Motion

To assess the model's performance in a more complex environment, we consider the KIT Whole-Body Human Motion Database (Mandery et al., 2016). This captures subject-specific motions, which are standardized based on the subject's height and weight using a reference kinematics and dynamics model known as the master motor map (MMM).

First, we focus on learning the motion in the 44-dimensional joint space of the human avatar. We chose the motion that corresponds to a human performing a tennis forehand skill. The architecture is the same as in Sec. 5.6.1 except that the Jacobian network was implemented as a neural network with two hidden layers, each containing 50 nodes. For this specific skill, we use a single demonstration to learn the skill, showing that NCDS is able to learn and generalize very complex skills with very few data. The results are shown in Fig. 6.14–*left*, where the motion progresses from left to right over time. The top row displays the demonstrated motion, the middle row shows the motion generated by a path integral starting from one of the demonstration's initial points, and the bottom row displays the motion generated by a path integral starting far from the demonstration's initial points. These results demonstrate that NCDS is

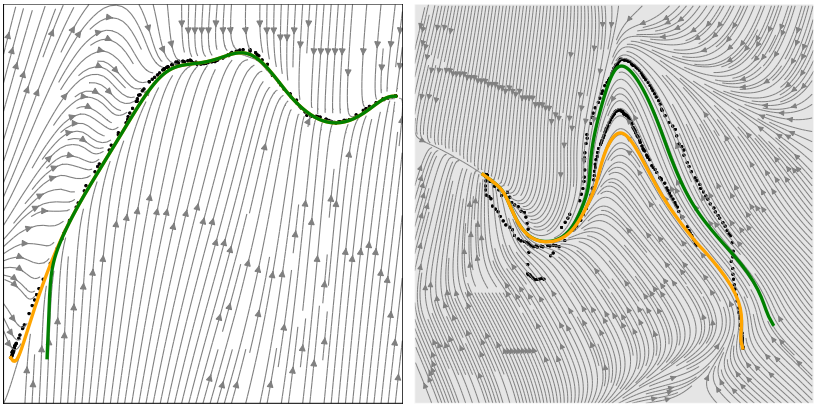


Figure 6.15: *Left:* Latent path integrals with NCDS trained in joint space $\mathbb{R}^{44}$. *Right:* Latent path integrals with NCDS trained on full human motion space: $\mathbb{R}^{44} \times \mathbb{R}^3 \times \mathcal{SO}(3)$. The background depicts the contours of the latent vector field, green and yellow curves depict the path integrals, and black dots represent demonstration in latent space.

able to learn high-dimensional dynamics. Figure 6.15–*left* illustrates the path integrals generated by NCDS on a 2D latent space.

The previous human motion experiment focused on reconstructing motions in the human joint space alone, thereby failing to model the global position and orientation of the human, which may be insufficient for some particular human motions. The KIT motion database also includes the base link pose of the human avatar, located at the hip. We have chosen a leg kick action to emphasize the importance of learning the base-link pose. Without incorporating the base-link pose and only focusing on joint motion, the movement appears unnatural, making the human avatar look as if it is floating or disconnected from the ground. Learning the base-link pose, which anchors the motion at the hip, helps maintain realistic contact with the ground, resulting in a more grounded and natural-looking movement. The architecture remained the same as in the joint-space experiment, except that the Jacobian network was implemented as a neural network with two hidden layers of 200 nodes each. For this skill, we used two demonstrations to learn the motion. Here each point of the motion trajectory is defined in a 44-dimensional joint space $\mathbb{R}^{44}$, along with the position and orientation of the base link $\mathbb{R}^3 \times \mathcal{SO}(3)$, leading to a 50-dimensional input space $\mathbb{R}^{44} \times \mathbb{R}^3 \times \mathcal{SO}(3)$.

Figure 6.15–*right* shows the latent path integrals generated by NCDS, while Fig. 6.14–*right* shows the corresponding motion. The upper row of the latter plot depicts the

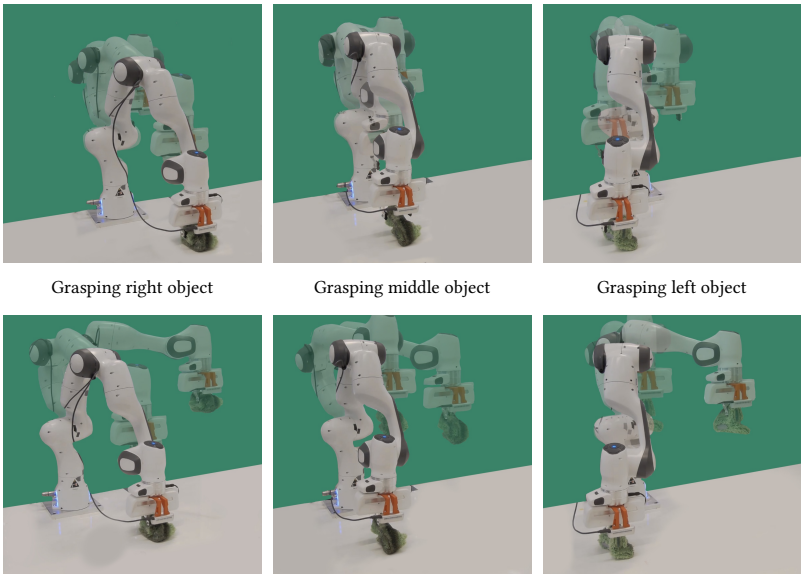


Figure 6.16: Grasping and dropping actions learned by the CNCDS for a soft object in three positions, conditioned on image input. The CNCDS is trained using cross-entropy loss, velocity reconstruction, and regularization with *ResNet18* image embeddings.

progressive evolution of the demonstration motion from left to right. Meanwhile, the middle and bottom rows illustrate the motions generated by NCDS when the initial point is respectively distant from and coincident with the initial point of the demonstration. Interestingly, we experimentally found that even this complex skill can be effectively learned using only a 2-dimensional latent space. This may be task-dependent, as some tasks might require a higher-dimensional latent space to be accurately encoded (Beik-Mohammadi et al., 2023). We have not observed this necessity in our experiments.

6.4.8 Vision-based Grasp-and-drop with CNCDS

The Conditional Neural Dynamical System (CNCDS) can learn multiple skills using a single network, which we demonstrate using a vision perception system. We place a soft object in three distinct positions on a table, and a camera captures an image of the

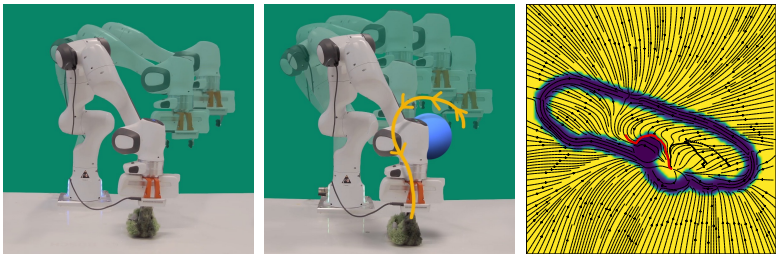


Figure 6.17: Obstacle avoidance via a Riemannian pullback metric. *Left*: The robot performs a grasping task without obstacle avoidance. *Middle*: The robot navigates around the obstacle by avoiding both the obstacle and the unsafe regions of the manifold. *Right*: The latent modulated vector field is locally reshaped around the obstacle and in areas outside the data support.

scene for each condition. We use a pre-trained ResNet18 model to generate a 1000-dimensional feature vector, which is then passed through a fully connected network to produce a 3-dimensional embedding vector that serves as the conditioning input. During training, only the ResNet18 model is kept frozen, while the fully connected network and the NCDS are trained end-to-end. The training optimizes a combined loss function comprising cross-entropy loss $\mathcal{L}_{\text{CE}}$, velocity reconstruction loss $\mathcal{L}_{\text{vel}}$, and regularization loss $\mathcal{L}_{\epsilon}$, as defined in the NCDS framework. The cross-entropy loss is given by $\mathcal{L}_{\text{CE}} = -\sum_i y_i \log(\hat{y}_i)$, where y_i represents the true label, and $\hat{y}_i$ is the predicted probability for class i. The fully connected layer consists of a single layer with 1000 nodes. The Jacobian network consists of a single hidden layer with 100 nodes, using a ReLU activation function. The VAE encoder/decoder is designed with two layers, containing 200 and 100 nodes, respectively.

Two separate CNCDS models are trained: one for grasping the object and another for dropping it, each functioning as an independent CNCDS conditioned on the initial image of the object on the table. For both the grasping and dropping skills, we collected 7 demonstrations, each containing 500 datapoints that capture the full trajectory of the motion. Additionally, for each object location relevant to the task, we gathered 50 initial snapshot images that represent the initial state of the object on the table. To build the complete dataset, each data point in the motion trajectories was paired with each of the initial snapshot images. The first three panels in Fig. 6.16 show the grasping actions from each initial position, while the second three panels show the corresponding dropping actions. By conditioning on visual information, the CNCDS framework effectively adapts to varying initial object positions, successfully learning and reproducing the desired skills.

6.5 Riemannian Safety Regions

In this section, we employ the pullback metric derived from the decoder's Jacobian to formulate the modulation of a contractive dynamical system. Specifically, the robot uses modulation to navigate around obstacles and avoid uncertain regions far from the demonstrations, under the assumption that out-of-support data regions are unsafe. As explained in Sec. 6.3, we use a Riemannian pullback metric computed via (2.13), to formulate the modulation matrix in (6.9). Specifically, we define the ambient metric $\boldsymbol{M}_{\mathcal{X}}$ for the end-effector position as in (3.17). Note that we use a Gaussian-like function to represent the obstacle here; however, more detailed representations, such as meshes constructed from point clouds, can also be employed. We demonstrate this approach by applying the dynamics learned from the grasping skill experiment reported in Sec. 6.4.8, where vector field modulation is incorporated based on the Riemannian formulation introduced in Sec. 6.3.

In Fig. 6.17, the *left* panel illustrates the robot performing a skill by grasping an object from the table without considering obstacle avoidance or designated safety regions. The *middle* panel displays the same action, this time with obstacle avoidance and safety region constraints enabled. The *right* panel represents the associated latent space, where darker areas represent unsafe zones. The dark oval-shaped region represents the Riemannian manifold derived from skill demonstrations, marking the boundary of safety regions within the latent space. The dark circular region inside represents the latent representation of the obstacle, obtained by pulling back the ambient metric into the latent space. These experiments highlight the potential of Riemannian modulation in enabling safe and adaptive robot behavior.

Summary and Outlook

By integrating geometry-aware constraints, the robot not only avoids obstacles but also respects learned safety regions, showcasing a promising direction for motion generation in uncertain environments. However, despite these encouraging results, important questions remain. In the next chapter, we discuss the current limitations of the presented methods, particularly their reliance on latent representations and the challenges of generalizing to out-of-distribution scenarios. We also identify several open challenges and outline potential avenues for extending the framework to more complex, real-world settings.

7 Limitations, Conclusion and Future Work

This chapter provides a high-level reflection on the broader limitations, insights, and future directions stemming from the research presented in this thesis. While the individual contributions have introduced novel tools and demonstrated their applicability across a range of robot learning problems, the goal here is to take a step back and assess the overarching themes and implications.

The thesis builds upon two foundational pillars: Riemannian geometry and contraction theory. By leveraging these concepts, the thesis offers a unified geometric perspective that brings together trajectory-based learning via geodesics and dynamical system-based learning through contraction theory, highlighting their complementary roles in enabling reactive and stable robot motion from demonstrations. However, this geometric formulation also introduces challenges. In particular, solving geodesic equations in high-dimensional, learned manifolds is computationally demanding and rarely analytically tractable. Similarly, while contraction-based dynamical systems offer strong theoretical guarantees, ensuring their scalability and interpretability in real-world scenarios remains nontrivial.

These insights reveal a core tension: while geometry provides structure, it also imposes constraints. Understanding and managing this trade-off is key to future progress. The remainder of this chapter discusses specific limitations tied to each component of the framework, and proposes directions that may help bridge the gap between theoretical elegance and practical deployment.

7.1 Geodesic Motion Skills

In this thesis, we first presented a novel learning-from-demonstration (LfD) framework that learns Riemannian manifolds from human demonstrations. These manifolds enable geodesic curves to be computed and utilized as motion generators for robot tasks. Specifically, the geodesics are computed as shortest paths on the learned manifold, leveraging the pullback Riemannian metric obtained through variational autoencoders

(VAEs). This approach facilitates motion planning between arbitrary points on the manifold, providing a versatile tool for robot motion generation in both task and joint spaces.

The geodesics synthesized using our method successfully recover and generate robot movements analogous to the provided demonstrations. To enhance real-time applicability, we developed a graph-based geodesic computation technique built on Dijkstra's algorithm. By discretizing the manifold into a graph representation, this approach significantly reduces computational overhead compared to direct numerical optimization methods for geodesic computation.

Additionally, we introduced a reshaping mechanism for the Riemannian metric using ambient metrics, enabling robots to avoid static and dynamic obstacles on the fly. The reshaped metric creates a potential field around obstacles, effectively pushing geodesics away from unsafe regions. Coupled with a forward kinematics layer, this allowed for multiple-limb obstacle avoidance and reactive motion capabilities.

We now turn our attention to the limitations of this approach and outline promising directions for future exploration.

Obstacle avoidance: Our obstacle avoidance strategy employed a "soft constraint" approach through ambient metrics, which theoretically might allow geodesics to cross obstacles under certain conditions. Although this behavior was not observed in our experiments, future work could integrate a "hard constraint" by removing nodes near obstacles in the graph representation. This method would guarantee obstacle-free paths while maintaining computational efficiency. Additionally, extending this framework to handle complex obstacle geometries, represented as point clouds, requires further research to maintain real-time performance.

Manifold generalization: When geodesics are forced to leave the learned manifold due to high-energy boundaries or obstacles, undesirable and inaccurate motions may occur. This issue arises from the VAE's latent space representation, where data outside the training distribution may be misrepresented. Future work could explore learning bijective mappings between old demonstrations and new task conditions or incorporating out-of-distribution detection mechanisms to improve robustness.

Latent space dimensionality and topology: The choice of latent space dimensionality directly impacts geodesic computation. For joint space tasks, higher-dimensional latent spaces are often required to capture the complexity of motion. This increases computational demands, particularly for calculating the Riemannian metric. Investigating

alternative latent space topologies, such as hyperspherical or hyperbolic spaces, could improve representational efficiency. Theoretical analysis of the latent space's curvature and its implications for geodesic smoothness is an important avenue for future research.

Integration of perceptual data: Our current framework encodes task- and joint-space trajectories without incorporating external sensory data. Expanding the latent space to include high-dimensional perceptual inputs, such as images, would enable richer motion representations. This requires addressing challenges in disentangled latent representations and efficient geodesic computation in high-dimensional spaces. Techniques like sparse Gaussian processes or neural approximation of geodesics could offer viable solutions.

Scalability to high-DOF Robots: As robots with higher degrees of freedom (DOF) are considered, computational efficiency becomes critical. Exploring gradient-based geodesic computation methods with parallelization or leveraging advances in graph neural networks for manifold representations could significantly enhance scalability.

7.2 Neural Contractive Dynamical Systems

While geodesics can effectively reconstruct complex motion skills, they fail to account for the temporal dynamics and timing of demonstrations, focusing solely on spatial aspects. To address this limitation, in the second part of the thesis, we introduce Neural Contractive Dynamical Systems (NCDS), which emphasize stability by leveraging contraction guarantees. By focusing on contraction, NCDS ensures robust and stable learning-based approaches for encoding dynamic motions, overcoming the challenges of uncontrolled extrapolation inherent in neural network models.

We employed state-independent and state-dependent regularization strategies to fine-tune contraction properties. Eigenvalue-based regularization methods ensured that the Jacobian's largest eigenvalue remained within the contraction bound, enhancing robustness and stability. These techniques provided better control over the system's generalization behavior, particularly in high-dimensional spaces.

To simplify the framework, we explored the effect of adding asymmetric components to the Jacobian matrix. Our results showed no significant performance gains from these additions, affirming that symmetric Jacobians are sufficient to locally approximate non-symmetric systems. This finding eliminates unnecessary complexity in the design.

By introducing conditional variables, the CNCDS framework adapted to multiple motion skills. The conditional variables influence the contraction metric, allowing a single NCDS module to handle diverse tasks. For example, target states were encoded as conditions, enabling the system to generalize across different goal configurations.

Inspired by modulation matrix techniques, we introduced Riemannian safety regions in the latent space. This approach reshapes the vector field near unsafe areas, ensuring contractive stability while guiding trajectories around obstacles. The modulation matrix enforces a local change in the contraction metric, dynamically altering the system's behavior in unsafe regions.

We now focus on the specific limitations of the proposed Neural Contractive Dynamical Systems (NCDS) framework and highlight potential directions for its refinement and future development.

Numerical integration sensitivity: Adaptive step sizing is essential for accurate numerical integration, particularly when computing trajectories in the latent space. However, this increases computational demands. Developing more efficient integration schemes, such as implicit integration methods, could address this limitation.

Computational overhead: Calculating the Jacobian of the decoder at each step for ambient velocity and Riemannian metric computations imposes significant overhead. Future research could explore methods to approximate these quantities, such as using precomputed Jacobians or neural network surrogates.

Reactivity to conditional variables: The CNCDS framework's ability to adapt to changes in conditional variables during skill execution remains limited. Investigating real-time reactivity mechanisms, such as incorporating time-dependent conditional variables or dynamic reweighting strategies, could enhance adaptability.

High-dimensional scaling: Scaling to high-dimensional latent spaces necessitates advancements in geodesic computation and uncertainty estimation. Techniques such as Riemannian variational autoencoders or neural ordinary differential equations (ODEs) could offer promising solutions.

7.3 Conclusion

The contributions of this thesis have advanced the state-of-the-art in motion generation through geodesic motion skills and Neural Contractive Dynamical Systems (NCDS).

These frameworks provide robust, adaptable, and theoretically sound solutions for robot motion generation in dynamic and complex environments. While challenges such as computational efficiency, scalability, and integration with sensory data remain, the proposed methodologies lay a strong foundation for future advancements in robot motion planning and control. Further exploration of latent space geometry, contraction properties, and high-dimensional scaling will drive the development of more sophisticated robotic systems.

Bibliography

Aljalbout, Elie; Chen, Ji; Ritt, Konstantin; Ulmer, Maximilian and Haddadin, Sami: "Learning Vision-based Reactive Policies for Obstacle Avoidance". In: *Conference on Robot Learning (CoRL)*. 2020. URL: https://proceedings.mlr.press/v155/aljalbout21a.html (cit. on p. 20).

Arvanitidis, Georgios; Hansen, Lars Kai and Hauberg, Søren: "Latent Space Oddity: on the Curvature of Deep Generative Models". In: *International Conference on Learning Representations (ICLR)*. 2018. URL: https://openreview.net/forum?id=SJzRZ-WCZ (cit. on pp. 13, 29, 38).

Arvanitidis, Georgios; Hauberg, Søren; Hennig, Philipp and Schober, Michael: "Fast and Robust Shortest Paths on Manifolds Learned from Data". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2019, pp. 1506–1515. URL: http://proceedings.mlr.press/v89/arvanitidis19a.html (cit. on pp. 31, 35).

Arvanitidis, Georgios; Hauberg, Søren and Schölkopf, Bernhard: "Geometrically Enriched Latent Spaces". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2021. URL: https://proceedings.mlr.press/v130/arvanitidis21a.html (cit. on p. 18).

Bahl, Shikhar; Mukadam, Mustafa; Gupta, Abhinav and Pathak, Deepak: "Neural Dynamic Policies for End-to-End Sensorimotor Learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020, pp. 5058–5069. URL: https://proceedings.neurips.cc/paper/2020/file/354ac345fd8c6d7ef634d9a8e3d47b83-Paper.pdf (cit. on p. 19).

Beik-Mohammadi, H.; Hauberg, S.; Arvanitidis, G.; Neumann, G. and Rozo, L.: "Learning Riemannian Manifolds for Geodesic Motion Skills". In: *Robotics: Science and Systems (R:SS)*. 2021. URL: https://roboticsconference.org/2021/program/papers/082/index.html (cit. on pp. 2, 8, 11, 25, 81).

Beik-Mohammadi, Hadi; Hauberg, Søren; Arvanitidis, Georgios; Figueroa, Nadia; Neumann, Gerhard and Rozo, Leonel: "Neural Contractive Dynamical Systems". In: *The Twelfth International Conference on Learning Representations*.

2024. URL: https://openreview.net/forum?id=iAYIRHOYy8 (cit. on pp. 6, 9, 11, 59, 81).

Beik-Mohammadi, Hadi; Hauberg, Søren; Arvanitidis, Georgios; Neumann, Gerhard and Rozo, Leonel: "Extended Neural Contractive Dynamical Systems: On Multiple Tasks and Riemannian Safety Regions". In: *The International Journal of Robotics Research (IJRR)* (2025). Accepted for publication. Copyright © 2025 SAGE Publications. URL: https://arxiv.org/abs/2411.11405 (cit. on pp. 8, 9, 11, 59, 81).

Beik-Mohammadi, Hadi; Hauberg, Søren; Arvanitidis, Georgios; Neumann, Gerhard and Rozo, Leonel: "Reactive motion generation on learned Riemannian manifolds". In: *The International Journal of Robotics Research (IJRR)* 42.10 (2023). Copyright © 2025 SAGE Publications., pp. 729–754. URL: https://doi.org/10.1177/02783649231193046 (cit. on pp. 3, 9, 11, 45, 108).

Billard, A.; Calinon, S. and Dillmann, R.: "Learning From Humans". In: *Handbook of Robotics*. Ed. by Siciliano, B. and Khatib, O. 2nd Edition. Secaucus, NJ, USA: Springer, 2016. Chap. 74, pp. 1995–2014. URL: https://doi.org/10.1007/978-3-319-32552-1_74 (cit. on p. 4).

Bishop, Christopher M; Svensén, Markus and Williams, Christopher KI: "Magnification factors for the SOM and GTM algorithms". In: *Workshop on Self-Organizing Maps*. 1997. URL: https://research.aston.ac.uk/files/113377/NCRG_97_008.pdf (cit. on p. 32).

Blocher, Caroline; Saveriano, Matteo and Lee, Dongheui: "Learning stable dynamical systems using contraction theory". In: *IEEE International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. 2017, pp. 124–129. URL: https://ieeexplore.ieee.org/document/7992901 (cit. on pp. 6, 95).

Brehmer, Johann and Cranmer, Kyle: "Flows for simultaneous manifold learning and density estimation". In: *Neural Information Processing Systems (NeurIPS*. 2020, pp. 442–453. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/051928341be67dcba03f0e04104d9047-Paper.pdf (cit. on pp. 16, 82, 100, 103).

Bullo, Francesco and Lewis, Andrew D.: Geometric Control of Mechanical Systems: Modeling, Analysis, and Design for Simple Mechanical Control Systems. Texts in Applied Mathematics. Springer New York, 2004. URL: https://books.google.de/books?id=jg7VumxOoe4C (cit. on p. 25).

Calinon, Sylvain: "A tutorial on task-parameterized movement learning and retrieval". In: *Intelligent Service Robotics* 9.1 (2016), pp. 1–29. URL: https://doi.org/10.1007/s11370-015-0187-9 (cit. on pp. 19, 20).

Campa, Ricardo and Camarillo, Karla: "Unit Quaternions: A Mathematical Tool for Modeling, Path Planning and Control of Robot Manipulators". In: *Robot Manipulators*. Rijeka: IntechOpen, 2008. Chap. 2. URL: https://doi.org/10.5772/6197 (cit. on p. 27).

Chen, Nutan; Ferroni, Francesco; Klushyn, Alexej; Paraschos, Alexandros; Bayer, Justin and Smagt, Patrick van der: "Fast Approximate Geodesics for Deep Generative Models". In: *International Conference on Artificial Neural Networks (ICANN)*. 2019, pp. 554–566. URL: https://link.springer.com/content/pdf/10.1007/978-3-030-30484-3_45.pdf (cit. on p. 31).

Chen, Nutan; Klushyn, Alexej; Kurle, Richard; Jiang, Xueyan; Bayer, Justin and Smagt, Patrick: "Metrics for Deep Generative Models". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2018, pp. 1540–1550. URL: http://proceedings.mlr.press/v84/chen18e.html (cit. on p. 31).

Chen, Nutan; Klushyn, Alexej; Paraschos, Alexandros; Benbouzid, Djalel and Smagt, Patrick Van der: "Active Learning based on Data Uncertainty and Model Sensitivity". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 1547–1554. URL: http://doi.org/10.1109/IROS.2018.8593552 (cit. on p. 81).

Chen, Ricky T. Q.: torchdiffeq. 2018. URL: https://github.com/rtqichen/torchdiffeq (cit. on p. 74).

Chen, Ricky T. Q.; Rubanova, Yulia; Bettencourt, Jesse and Duvenaud, David K: "Neural Ordinary Differential Equations". In: *Neural Information Processing Systems (NeurIPS)*. 2018. URL: https://proceedings.neurips.cc/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf (cit. on p. 61).

Cormen, Thomas H; Leiserson, Charles E; Rivest, Ronald L and Stein, Clifford: Introduction to algorithms. MIT press, 2009. URL: https://mitpress.mit.edu/books/introduction-algorithms-third-edition (cit. on p. 36).

Dawson, Charles; Gao, Sicun and Fan, Chuchu: "Safe Control With Learned Certificates: A Survey of Neural Lyapunov, Barrier, and Contraction Methods for Robotics and Control". In: *IEEE Transactions on Robotics (T-RO)* 39.3 (2023), pp. 1749–1767. URL: https://doi.org/10.1109/TRO.2022.3232542 (cit. on pp. 6, 7, 22, 95).

Deisenroth, Marc Peter; Faisal, A. Aldo and Ong, Cheng Soon: Mathematics for Machine Learning. Cambridge University Press, 2020. URL: https://doi.org/10.1017/9781108679930 (cit. on p. 46).

Dollár, Piotr; Rabaud, Vincent and Belongie, Serge: "Non-Isometric Manifold Learning: Analysis and an Algorithm". In: *International Conference on Machine Learning (ICML)*. 2007, pp. 241–248. doi: 10.1145/1273496.1273527. url: https://pdollar.github.io/files/papers/DollarICML07manifold.pdf (cit. on pp. 2, 25).

Eklund, David and Hauberg, Søren: "Expected path length on random manifolds". In: *arXiv preprint*. 2019. url: https://arxiv.org/abs/1908.07377 (cit. on p. 17).

Elbanhawi, Mohamed and Simic, Milan: "Sampling-Based Robot Motion Planning: A Review". In: *IEEE Access* 2 (2014), pp. 56–77. url: https://ieeexplore.ieee.org/document/6722915 (cit. on pp. 2, 19, 49).

Ewerton, Marco; Neumann, Gerhard; Lioutikov, Rudolf; Ben Amor, Heni; Peters, Jan and Maeda, Guilherme: "Learning multiple collaborative tasks with a mixture of interaction Primitives". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 1535–1542. url: https://doi.org/10.1109/ICRA.2015.7139393 (cit. on p. 20).

Falorsi, Luca; de Haan, Pim; Davidson, Tim R. and Forré, Patrick: "Reparameterizing Distributions on Lie Groups". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2019, pp. 3244–3253. url: https://proceedings.mlr.press/v89/falorsi19a.html (cit. on p. 86).

Giesl, Peter and Hafstein, Sigurdur: "Review on computational methods for Lyapunov functions". In: *Discrete and Continuous Dynamical Systems - B* 20.8 (2015), pp. 2291–2331. doi: 10.3934/dcdsb.2015.20.2291. url: https://www.aimsciences.org/article/id/ee7e4111-9a13-4e44-95c8-14c5557278fc (cit. on p. 21).

Hauberg, Søren; Feragen, Aasa; Enficiaud, Raffi and Black, Michael J.: "Scalable Robust Principal Component Analysis using Grassmann Averages". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 38.11 (2016), pp. 2298–2311. url: https://doi.org/10.1109/TPAMI.2015.2511743 (cit. on p. 28).

Hauberg, Søren: Only Bayes should learn a manifold. 2019. url: https://arxiv.org/abs/1806.04994 (cit. on p. 17).

Havoutis, Ioannis and Ramamoorthy, Subramanian: "Motion planning and reactive control on learnt skill manifolds". In: *International Journal of Robotics Research (IJRR)* 32.9-10 (2013), pp. 1120–1150. doi: 10.1177/0278364913482016. url: https://doi.org/10.1177/0278364913482016 (cit. on pp. 2, 25).

Hemingway, Evan G. and O'Reilly, Oliver M.: "Perspectives on Euler angle singularities, gimbal lock, and the orthogonality of applied forces and applied moments". In: *Multibody System Dynamics* 44.1 (2018), pp. 31–56. doi: 10.1007/s11044-018-9620-0. url: https://doi.org/10.1007/s11044-018-9620-0 (cit. on p. 27).

Higgins, Irina; Matthey, Loïc; Pal, Arka; Burgess, Christopher P.; Glorot, Xavier; Botvinick, Matthew M.; Mohamed, Shakir and Lerchner, Alexander: "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In: *International Conference on Learning Representations*. 2017. url: https://openreview.net/forum?id=Sy2fzU9gl (cit. on p. 28).

Huang, Yanlong; Rozo, Leonel; Silvério, João and Caldwell, Darwin G: "Kernelized movement primitives". In: *International Journal of Robotics Research (IJRR)* 38.7 (2019), pp. 833–852. url: https://doi.org/10.1177/0278364919846363 (cit. on pp. 19, 20).

Huber, Lukas; Billard, Aude and Slotine, Jean-Jacques: "Avoidance of Convex and Concave Obstacles With Convergence Ensured Through Contraction". In: *IEEE Robotics and Automation Letters (RA-L)* 4.2 (2019), pp. 1462–1469. doi: http://doi.org/10.1109/LRA.2019.2893676 (cit. on pp. 22, 23, 72).

Huber, Lukas; Slotine, Jean-Jacques and Billard, Aude: "Avoiding Dense and Dynamic Obstacles in Enclosed Spaces: Application to Moving in Crowds". In: *IEEE Transactions on Robotics* 38.5 (2022), pp. 3113–3132. url: https://ieeexplore.ieee.org/document/9765824 (cit. on pp. 23, 70–72, 89).

Hung, Chia-Man; Zhong, Shaohong; Goodwin, Walter; Jones, Oiwi Parker; Engelcke, Martin; Havoutis, Ioannis and Posner, Ingmar: "Reaching Through Latent Space: From Joint Statistics to Path Planning in Manipulation". In: *IEEE Robotics and Automation Letters (RA-L)* 7.2 (2022), pp. 5334–5341. url: http://doi.org/10.1109/LRA.2022.3152697 (cit. on p. 81).

Ijspeert, Auke Jan; Nakanishi, Jun; Hoffmann, Heiko; Pastor, Peter and Schaal, Stefan: "Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors". In: *Neural Computation* 25 (2013), pp. 328–373. url: https://homes.cs.washington.edu/~todorov/courses/amath579/reading/DynamicPrimitives.pdf (cit. on p. 19).

Jaffe, Sean; Davydov, Alexander; Lapsekili, Deniz; Singh, Ambuj and Bullo, Francesco: Learning Neural Contracting Dynamics: Extended Linearization and Global Guarantees. 2024. url: https://arxiv.org/abs/2402.08090 (cit. on pp. 67, 72, 95, 98).

Jaquier, Noémie; Rozo, Leonel; Calinon, Sylvain and Bürger, Mathias: "Bayesian Optimization Meets Riemannian Manifolds in Robot Learning". In: *Conference on Robot Learning (CoRL)*. 2020, pp. 233–246. url: http://proceedings.mlr.press/v100/jaquier20a.html (cit. on p. 27).

Jouffroy, Jerome and I. Fossen, Thor: "A Tutorial on incremental stability analysis using contraction theory". In: *Modeling, Identification and Control* (2010). url: https://doi.org/10.4173/mic.2010.3.2 (cit. on p. 21).

Kalatzis, Dimitris; Eklund, David; Arvanitidis, Georgios and Hauberg, Søren: "Variational Autoencoders with Riemannian Brownian Motion Priors". In: *International Conference on Machine Learning (ICML)*. Vol. 119. 2020, pp. 6789–6799. url: http://proceedings.mlr.press/v119/kalatzis20a/kalatzis20a.pdf (cit. on p. 31).

Khansari-Zadeh, S. Mohammad and Billard, Aude: "Learning control Lyapunov function to ensure stability of dynamical system-based robot reaching motions". In: *Robotics and Autonomous Systems (RAS)* 62.6 (2014), pp. 752–765. doi: https://doi.org/10.1016/j.robot.2014.03.001 (cit. on pp. 4, 5).

Khansari-Zadeh, S. Mohammad and Billard, Aude: "Learning Stable Nonlinear Dynamical Systems With Gaussian Mixture Models". In: *IEEE Transactions on Robotics* 27.5 (2011), pp. 943–957. doi: 10.1109/TRO.2011.2159412 (cit. on pp. 4, 5, 96).

Khansari-Zadeh, S. Mohammad and Billard, Aude: "Learning Stable Nonlinear Dynamical Systems With Gaussian Mixture Models". In: *IEEE Transactions on Robotics* 27.5 (2011), pp. 943–957. url: https://ieeexplore.ieee.org/document/5953529 (cit. on p. 19).

Kingma, Diederik P. and Ba, Jimmy: "Adam: A Method for Stochastic Optimization". In: *CoRR* (2014). url: https://api.semanticscholar.org/CorpusID:6628106 (cit. on p. 76).

Kingma, Diederik P. and Welling, Max: "Auto-Encoding Variational Bayes". In: *International Conference on Learning Representations (ICLR)*. 2014. url: https://openreview.net/forum?id=33X9fd2-9FyZd (cit. on pp. 15, 17, 20).

Kirk, D.E.: Optimal Control Theory: An Introduction. Networks series. Prentice-Hall, 1970. url: https://store.doverpublications.com/0486434842.html (cit. on p. 25).

Konidaris, George; Kuindersma, Scott; Grupen, Roderic and Barto, Andrew: "Robot learning from demonstration by constructing skill trees". In: *The International Journal of Robotics Research (IJRR)* 31.3 (2012), pp. 360–375. URL: https://doi.org/10.1177/0278364911428653 (cit. on p. 20).

Koptev, Mikhail: "Implicit Distance Functions: Learning and Applications in Robotics". In: (2023). DOI: https://doi.org/10.5075/epfl-thesis-10197. URL: http://infoscience.epfl.ch/record/305181 (cit. on pp. 90, 92).

Kozachkov, Leo; Wensing, Patrick and Slotine, Jean-Jacques: "Generalization as Dynamical Robustness–The Role of Riemannian Contraction in Supervised Learning". In: *Transactions on Machine Learning Research* (2023). URL: https://openreview.net/forum?id=Sb6p5mcefw (cit. on p. 82).

Lee, John: Introduction to Riemannian Manifolds. 2nd ed. Springer, 2018. URL: https://sites.math.washington.edu/~lee/Books/RM/ (cit. on p. 12).

Lemme, A.; Meirovitch, Y.; Khansari-Zadeh, M.; Flash, T.; Billard, A. and Steil, J. J.: "Open-source benchmarking for learned reaching motion generation in robotics". In: *Paladyn, Journal of Behavioral Robotics* 6.1 (2015). URL: http://doi.org/10.1515/pjbr-2015-0002 (cit. on p. 74).

Li, Mao; Tahara, Kenji and Billard, Aude: "Learning task manifolds for constrained object manipulation". In: *Autonomous Robots* 42 (2018), pp. 159–174. URL: https://doi.org/10.1007/s10514-017-9643-z (cit. on pp. 2, 25).

Lohmiller, Winfried and Slotine, Jean-Jacques E.: "On Contraction Analysis for Non-linear Systems". In: *Automatica* 34.6 (1998), pp. 683–696. URL: https://doi.org/10.1016/S0005-1098(98)00019-3 (cit. on pp. 4, 5, 21, 22).

Lorraine, Jonathan and Hossain, Safwan: "JacNet: Learning Functions with Structured Jacobians". In: *INNF Workshop at the International Conference on Machine Learning (ICML)* (2019). URL: https://invertibleworkshop.github.io/INNF_2019/accepted_papers/pdfs/INNF_2019_paper_10.pdf (cit. on p. 60).

Manchester, Ian R. and Slotine, Jean-Jacques E.: "Control Contraction Metrics: Convex and Intrinsic Criteria for Nonlinear Feedback Design". In: *IEEE Transactions on Automatic Control* 62.6 (2017), pp. 3046–3053. URL: https://doi.org/10.1109/TAC.2017.2668380 (cit. on p. 82).

Mandery, Christian; Terlemez, Ömer; Do, Martin; Vahrenkamp, Nikolaus and Asfour, Tamim: "Unifying Representations and Large-Scale Whole-Body Motion Databases for Studying Human Motion". In: *IEEE Transactions on Robotics* 32.4 (2016), pp. 796–809. URL: https://ieeexplore.ieee.org/document/7506114 (cit. on pp. 93, 106).

Manschitz, Simon; Gienger, Michael; Kober, Jens and Peters, Jan: "Mixture of Attractors: A Novel Movement Primitive Representation for Learning Motor Skills From Demonstrations". In: *IEEE Robotics and Automation Letters (RA-L)* 3.2 (2018), pp. 926–933. URL: https://ieeexplore.ieee.org/document/8255585 (cit. on p. 19).

Mohanan, M.G. and Salgoankar, Ambuja: "A survey of robotic motion planning in dynamic environments". In: *Robotics and Autonomous Systems* 100 (2018), pp. 171–185. URL: https://www.sciencedirect.com/science/article/pii/S0921889017300313 (cit. on p. 2).

Neumann, Klaus and Steil, Jochen J.: "Learning robot motions with stable dynamical systems under diffeomorphic transformations". In: *Robotics and Autonomous Systems (RAS)* 70 (2015), pp. 1–15. URL: https://doi.org/10.1016/j.robot.2015.04.006 (cit. on pp. 4, 5).

Noakes, Lyle and Zhang, Erchuan: "Finding geodesics joining given points". In: *Advances in Computational Mathematics* 48.4 (2022). URL: https://rdcu.be/dba9z (cit. on p. 36).

Norcliffe, Alexander Luke Ian; Bodnar, Cristian; Day, Ben; Simidjievski, Nikola and Lio, Pietro: "On Second Order Behaviour in Augmented Neural ODEs: A Short Summary". In: *Workshop on the Symbiosis of Deep Learning and Differential Equations at NeurIPS*. 2021. URL: https://openreview.net/forum?id=XpmaGtI04ki (cit. on p. 61).

Orocos: orocos/orocos_kinematics_dynamics: Orocos Kinematics and Dynamics C library. 2021. URL: https://github.com/orocos/orocos_kinematics_dynamics (cit. on p. 50).

Osa, Takayuki and Ikemoto, Shuhei: "Variational Autoencoder Trajectory Primitives with Continuous and Discrete Latent Codes". In: *CoRR* abs/1912.04063 (2019). URL: https://link.springer.com/article/10.1007/s42979-020-00324-7 (cit. on p. 19).

Paraschos, Alexandros; Daniel, Christian; Peters, Jan and Neumann, Gerhard: "Using probabilistic movement primitives in robotics". In: *Autonomous Robots* 42 (2018), pp. 529–551. URL: https://link.springer.com/article/10.1007/s10514-017-9648-7 (cit. on pp. 19, 20).

Paszke, Adam et al.: "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019, pp. 8024–8035. URL: https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf (cit. on p. 38).

Paszke, Adam et al.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. 2019. URL: https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf (cit. on pp. 100, 103).

Peyré, Gabriel; Péchaud, Mickael; Keriven, Renaud and Cohen, Laurent D.: "Geodesic Methods in Computer Vision and Graphics". In: *Foundations and Trends in Computer Graphics and Vision* 5.3–4 (2010), pp. 197–397. URL: http://dx.doi.org/10.1561/0600000029 (cit. on p. 35).

Poli, Michael; Massaroli, Stefano; Yamashita, Atsushi; Asama, Hajime; Park, Jinkyoo and Ermon, Stefano: "TorchDyn: Implicit Models and Neural Numerical Methods in PyTorch". In: (2021). URL: https://github.com/DiffEqML/torchdyn/ (cit. on p. 76).

Prescott, Tony and Mayhew, John: "Obstacle Avoidance through Reinforcement Learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 4. Morgan-Kaufmann, 1992. URL: https://proceedings.neurips.cc/paper/1991/file/9431c87f273e507e6040fcb07dcb4509-Paper.pdf (cit. on p. 20).

Rana, Muhammad Asif; Li, Anqi; Fox, Dieter; Boots, Byron; Ramos, Fabio and Ratliff, Nathan: "Euclideanizing Flows: Diffeomorphic Reduction for Learning Stable Dynamical Systems". In: *Conference on Learning for Dynamics and Control (L4DC)*. 2020, pp. 630–639. URL: https://proceedings.mlr.press/v120/rana20a.html (cit. on p. 96).

Rana, Muhammad Asif; Li, Anqi; Fox, Dieter; Boots, Byron; Ramos, Fabio and Ratliff, Nathan: "Euclideanizing Flows: Diffeomorphic Reduction for Learning Stable Dynamical Systems". In: *Proceedings of the 2nd Conference on Learning for Dynamics and Control*. Vol. 120. Proceedings of Machine Learning Research. PMLR, 2020, pp. 630–639. URL: https://proceedings.mlr.press/v120/rana20a.html (cit. on pp. 19, 21).

Ratliff, Nathan; Zucker, Matthew; Bagnell, J. Andrew (Drew) and Srinivasa, Siddhartha: "CHOMP: Gradient Optimization Techniques for Efficient Motion Planning". In: *International Conference on Robotics and Automation (ICRA)*. 2009, pp. 489–494. URL: https://doi.org/10.1109/ROBOT.2009.5152817 (cit. on pp. 20, 34).

Ratliff, Nathan D.; Issac, Jan; Kappler, Daniel; Birchfield, Stan and Fox, Dieter: Riemannian Motion Policies. 2018. URL: https://arxiv.org/abs/1801.02854 (cit. on p. 20).

Ravichandar, Harish; Polydoros, Athanasios S; Chernova, Sonia and Billard, Aude: "Recent Advances in Robot Learning from Demonstration". In: *Annual Review of Control, Robotics, and Autonomous Systems* 3.1 (2020), pp. 297–330. URL: https://doi.org/10.1146/annurev-control-100819-063206 (cit. on p. 19).

Ravichandar, Harish Chaandar and Dani, Ashwin: "Learning Position and Orientation Dynamics from Demonstrations via Contraction Analysis". In: *Autonomous Robots* 43.4 (2019), 897–912. URL: https://doi.org/10.1007/s10514-018-9758-x (cit. on pp. 5, 95).

Ravichandar, Harish Chaandar; Salehi, Iman and Dani, Ashwin: "Learning Partially Contracting Dynamical Systems from Demonstrations". In: *Conference on Robot Learning (CoRL)*. 2017, pp. 369–378. URL: https://proceedings.mlr.press/v78/ravichandar17a.html (cit. on pp. 5, 94, 95).

Rezazadeh, Navid; Kolarich, Maxwell; Kia, Solmaz S. and Mehr, Negar: "Learning Contraction Policies From Offline Data". In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 2905–2912. DOI: 10.1109/LRA.2022.3145100 (cit. on p. 95).

Rozo, Leonel and Dave, Vedant: "Orientation Probabilistic Movement Primitives on Riemannian Manifold". In: *Conference on Robot Learning (CoRL)*. 2021. URL: https://proceedings.mlr.press/v164/rozo22a.html (cit. on p. 20).

Rozo, Leonel; Jimenez, Pablo and Torras, Carme: "Robot learning from demonstration of force-based tasks with multiple solution trajectories". In: *International Conference on Advanced Robotics (ICAR)*. 2011, pp. 124–129. URL: https://doi.org/10.1109/ICAR.2011.6088633 (cit. on p. 20).

Rozo, Leonel et al.: "Learning and Sequencing of Object-Centric Manipulation Skills for Industrial Tasks". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 9072–9079. URL: https://doi.org/10.1109/IROS45743.2020.9341570 (cit. on p. 27).

Schaal, Stefan; Ijspeert, Auke and Billard, Aude: "Computational approaches to motor learning by imitation". In: *Phil. Trans. R. Soc. Lond. B* 358 (2003), pp. 537–547. URL: http://doi.org/10.1098/rstb.2002.1258 (cit. on pp. 2, 4).

Seker, M. Yunus; Imre, Mert; Piater, Justus H. and Ugur, Emre: "Conditional Neural Movement Primitives". In: *Robotics: Science and Systems (R:SS)*. 2019. URL: http://www.roboticsproceedings.org/rss15/p71.html (cit. on pp. 19, 20).

Shao, Hang; Kumar, Abhishek and Fletcher, P. Thomas: "The Riemannian Geometry of Deep Generative Models". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2018, pp. 428–4288. URL: https://doi.org/10.1109/CVPRW.2018.00071 (cit. on p. 31).

Shuster, Malcolm David: "A survey of attitude representation". In: *Journal of The Astronautical Sciences* 41 (1993), pp. 439–517 (cit. on p. 85).

Sindhwani, Vikas; Tu, Stephen and Khansari, Mohi: "Learning Contracting Vector Fields For Stable Imitation Learning". In: *arXiv* 1804.04878 (2018). URL: https://arxiv.org/abs/1804.04878 (cit. on pp. 6, 94, 95).

Singh, Sumeet; Richards, Spencer M; Sindhwani, Vikas; Slotine, Jean-Jacques E and Pavone, Marco: "Learning stabilizable nonlinear dynamics with contraction-based regularization". In: *The International Journal of Robotics Research* 40.10-11 (2021), pp. 1123–1150. DOI: 10.1177/0278364920949931. URL: https://doi.org/10.1177/0278364920949931 (cit. on p. 95).

Solà, Joan; Deray, Jérémie and Atchuthan, Dinesh: "A micro Lie theory for state estimation in robotics". In: *CoRR* abs/1812.01537 (2018). arXiv: 1812.01537. URL: http://arxiv.org/abs/1812.01537 (cit. on p. 85).

Sra, Suvrit: "Directional Statistics in Machine Learning: A Brief Review". In: *Applied Directional Statistics*. Ed. by Ley, Christophe and Verdebout, Thomas. 1st edition. 2018. URL: https://arxiv.org/abs/1605.00316 (cit. on pp. 27, 28).

Sun, Dawei; Jha, Susmit and Fan, Chuchu: "Learning Certified Control using Contraction Metric". In: *Conference on Robot Learning (CoRL)*. 2020. URL: http://arxiv.org/abs/2011.12569 (cit. on pp. 6, 95).

Tabak, Esteban G and Turner, Cristina V: "A family of nonparametric density estimation algorithms". In: *Communications on Pure and Applied Mathematics* 66.2 (2013), pp. 145–164 (cit. on p. 16).

Tsukamoto, Hiroyasu and Chung, Soon Jo: "Neural Contraction Metrics for Robust Estimation and Control: A Convex Optimization Approach". In: *IEEE Control Systems Letters* 5 (1 2021), pp. 211–216. URL: https://doi.org/10.1109/LCSYS.2020.3001646 (cit. on p. 6).

Tsukamoto, Hiroyasu and Chung, Soon Jo: "Neural Contraction Metrics for Robust Estimation and Control: A Convex Optimization Approach". In: *IEEE Control Systems Letters* 5 (1 2021), pp. 211–216. URL: https://doi.org/10.1109/LCSYS.2020.3001646 (cit. on pp. 72, 95).

Tsukamoto, Hiroyasu and Chung, Soon-Jo: "Robust Controller Design for Stochastic Nonlinear Systems via Convex Optimization". In: *IEEE Transactions on Automatic Control* 66.10 (2021), pp. 4731–4746. doi: 10.1109/TAC.2020.3038402 (cit. on p. 72).

Tsukamoto, Hiroyasu; Chung, Soon-Jo; Slotine, Jean-Jacques and Fan, Chuchu: "A Theoretical Overview of Neural Contraction Metrics for Learning-based Control with Guaranteed Stability". In: *IEEE Conference on Decision and Control (CDC)*. 2021, pp. 2949–2954. url: https://ieeexplore.ieee.org/document/9682859 (cit. on p. 6).

Tsukamoto, Hiroyasu; Chung, Soon-Jo and Slotine, Jean-Jaques E.: "Contraction theory for nonlinear stability analysis and learning-based control: A tutorial overview". In: *Annual Reviews in Control* 52 (2021), pp. 135–169. url: https://doi.org/10.1016/j.arcontrol.2021.10.001 (cit. on pp. 21, 22).

Ugur, Emre and Girgin, Hakan: "Compliant Parametric Dynamic Movement Primitives". In: *Robotica* 38.3 (2020), 457–474. url: https://www.cmpe.boun.edu.tr/~emre/papers/Ugur-2019-ROB.pdf (cit. on p. 19).

Urain, J.; Anqi, L.; Puze, L.; D'eramo, C. and Peters, J.: "Composable Energy Policies for Reactive Motion Generation and Reinforcement Learning". In: *Robotics: Science and Systems (R:SS)*. 2021. url: http://www.roboticsproceedings.org/rss17/p052.html (cit. on p. 20).

Urain, J.; Ginesi, M.; Tateo, D. and Peters, J.: "ImitationFlow: Learning Deep Stable Stochastic Dynamic Systems by Normalizing Flows". In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 5231–5237. url: http://ras.papercept.net/images/temp/IROS/files/1340.pdf (cit. on pp. 21, 96).

Urain, Julen; Tateo, Davide and Peters, Jan: "Learning Stable Vector Fields on Lie Groups". In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 12569–12576. doi: 10.1109/LRA.2022.3219019. url: https://ieeexplore.ieee.org/document/9935105 (cit. on p. 86).

Xu, Keyulu; Zhang, Mozhi; Li, Jingling; Du, Simon Shaolei; Kawarabayashi, Ken-Ichi and Jegelka, Stefanie: "How Neural Networks Extrapolate: From Feedforward to Graph Neural Networks". In: *International Conference on Learning Representations (ICLR)*. 2021. url: https://openreview.net/forum?id=UH-cmocLJC (cit. on p. 5).

Zeestraten, Martijn: "Programming by Demonstration on Riemannian Manifolds". PhD thesis. University of Genova, Italy, 2018. url: https://iris.unige.it/retrieve/handle/11567/930621/245803/phdunige_4054447.pdf (cit. on p. 20).

Zhang, Jiechao; Beik-Mohammadi, Hadi and Rozo, Leonel: "Learning Riemannian Stable Dynamical Systems via Diffeomorphisms". In: *Conference on Robot Learning (CoRL)*. 2022. URL: https://openreview.net/pdf?id=o8dLx8OVcNk (cit. on pp. 4, 5, 21).

Zhang, Yu; Cheng, Long; Li, Houcheng and Cao, Ran: "Learning Accurate and Stable Point-to-Point Motions: A Dynamic System Approach". In: *IEEE Robotics and Automation Letters (RA-L)* 7.2 (2022), pp. 1510–1517. URL: https://doi.org/10.1109/LRA.2022.3140677 (cit. on pp. 4, 5).

List of Figures

List of Tables