Journal Title
XX(X):1–23
©The Author(s) 2016
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Extended Neural Contractive Dynamical Systems: On Multiple Tasks and Riemannian Safety Regions

Hadi Beik-Mohammadi[1 2], Søren Hauberg[3], Georgios Arvanitidis[3], Gerhard Neumann[2], and Leonel Rozo[1]

Abstract

Stability guarantees are crucial when ensuring that a fully autonomous robot does not take undesirable or potentially harmful actions. We recently proposed the *Neural Contractive Dynamical Systems (NCDS)*, which is a neural network architecture that guarantees contractive stability. With this, learning-from-demonstrations approaches can trivially provide stability guarantees. However, our early work left several unanswered questions, which we here address. Beyond providing an in-depth explanation of NCDS, this paper extends the framework with more careful regularization, a conditional variant of the framework for handling multiple tasks, and an uncertainty-driven approach to latent obstacle avoidance. Experiments verify that the developed system has the flexibility of ordinary neural networks while providing the stability guarantees needed for autonomous robotics.

Keywords

Neural Contraction, Stable Dynamical Systems, Robot Motion Skills

1 Introduction

Autonomous robotic systems require stability guarantees to ensure safe and reliable operation, especially in dynamic and unpredictable environments. Manually designed robot movements can achieve such stability guarantees, but even skilled engineers struggle to hand-code highly dynamic motions (Billard et al. 2016). In contrast, learning robot skills from human demonstrations is an efficient and intuitive approach for encoding highly dynamic motions into a robot's repertoire (Schaal et al. 2003). Unfortunately, learning-based approaches often struggle to ensure stability as they rely on the machine learning model to extrapolate in a controlled manner. In particular, controlling the extrapolation behavior of neural networks has proven challenging (Xu et al. 2021), which hampers stability guarantees.

Many tasks require the robot to dynamically follow desired trajectories, e.g. in flexible manufacturing, human-robot interaction, or entertainment settings. In these cases, asymptotic stability, which guarantees stability only with respect to a fixed point (Rana et al. 2020a; Khansari-Zadeh and Billard 2011; Zhang et al. 2022), is insufficient, demanding a more general stability concept. *Contraction theory* (Lohmiller and Slotine 1998; Bullo 2024) is particularly well-suited, as it ensures that all 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.

To address this problem, our previous work introduced Neural Contractive Dynamical Systems (NCDS, Beik-Mohammadi et al. (2024)), a neural network architecture designed to ensure contractive behavior across all parameter values. This model enables robots to maintain stability even when subjected to external perturbations. The original NCDS ("vanilla NCDS") framework provides a solid foundation for learning complex contractive dynamical systems. Still, there remains potential for further refinement, particularly in formulating the Jacobian of the learned system dynamics, which is crucial for stability and generalization capabilities. Vanilla NCDS employs a symmetric Jacobian with a constant regularization term to enforce contraction (Lohmiller and Slotine 1998; Jouffroy and I. Fossen 2010). While effective, this approach left room for exploring how different regularization methods and Jacobian formulations could enhance the model's performance both within and beyond the data support region.

The vanilla NCDS framework learns high-dimensional dynamics via a low-dimensional contractive latent space. However, it performs all obstacle avoidance computations in the ambient (high-dimensional) space, which may be computationally intensive. To the best of our knowledge, no existing method for learning dynamical systems can simultaneously learn contractive stable dynamics while also performing obstacle avoidance in the latent space. To achieve this, we extend the work of Beik-Mohammadi et al. (2023), and leverage a Riemannian pullback metric, derived from a VAE injective decoder, to lift the obstacle avoidance problem to the latent space. This ensures the avoidance of unsafe regions outside of data support. On the multi-task front, the vanilla NCDS framework, like most methods that learn contractive

[1] Bosch Center for Artificial Intelligence (BCAI), Renningen, Germany
[2] Autonomous Learning Robots Lab, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
[3] Section for Cognitive Systems, Technical University of Denmark (DTU), Lyngby, Denmark

Corresponding author:
Hadi Beik-Mohammadi, Bosch Center for Artificial Intelligence (BCAI)
Email: hadi.beik-mohammadi@de.bosch.com

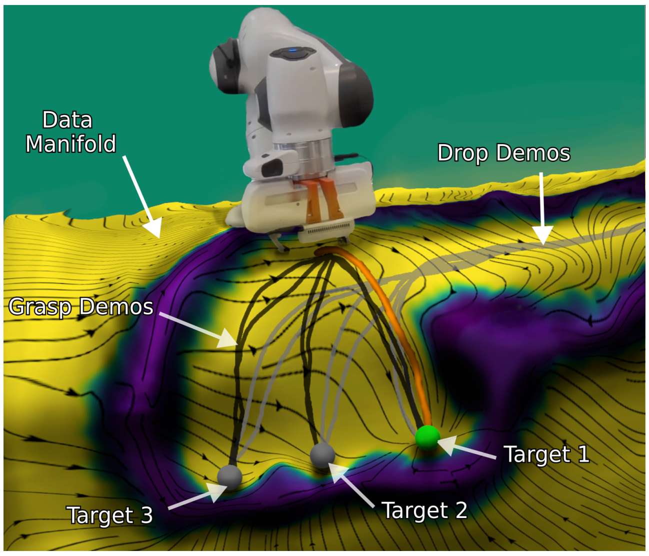


Figure 1. Robot motion generated for a grasping skill using a Conditional Neural Contractive Dynamical System (CNCDS). The robot motion is conditioned on an input image of the object placed on a desk. CNCDS also learns a Riemannian manifold from demonstrations, representing a safety region to navigate through.

dynamical systems (Tsukamoto and Chung 2021a; Dawson et al. 2023; Jaffe et al. 2024), was designed to learn a single skill. Although this may suffice for certain tasks, a more general approach would involve enabling the learning of multiple skills within the same contractive dynamical system. This reduces not only the use of computational resources but also the need to train separate models for each skill.

In this paper, we introduce several advancements to the vanilla NCDS aimed at improving the model's flexibility and robustness. Our **first contribution** focuses on the regularization of the Jacobian of the learned system dynamics. We investigate various unexplored strategies, including state-independent and state-dependent regularization vectors, as well as eigenvalue-based approach. These methods aim to fine-tune the contraction properties of the system, leading to more robust generalization and faster convergence within the data region. Our **second contribution** explores the potential benefits of introducing asymmetry into the system Jacobian matrix by incorporating both symmetric and skew-symmetric components. This asymmetry aims at increasing the model's flexibility, allowing it to capture more complex dynamical behaviors (Jaffe et al. 2024). This investigation builds on the understanding that, while symmetry in the Jacobian provides certain stability benefits, asymmetry could offer more nuanced control over the system's dynamics, particularly relevant in complex dynamic skills. Our **third contribution**, "conditional NCDS" (CNCDS), extends the model to handle multiple motion skills by conditioning on task-related variables such as target states. This conditional framework allows a single NCDS module to adapt to varying task conditions, broadening its applicability in more complex, multimodal robotic tasks. Finally, our **fourth contribution** extends NCDS with latent obstacle avoidance capabilities. Our approach, inspired by modulation matrix techniques, allows the system to navigate around dynamic obstacles while maintaining contractive stability (Huber et al. 2022, 2019).

We frame obstacle avoidance from a Riemannian perspective and define safety regions in the latent space through Riemannian pullback metrics (Beik-Mohammadi et al. 2023). This designates both obstacles and out-of-data-support regions as unsafe.

2 Learning contractive vector fields

In this section, we revisit the neural contractive dynamical system (NCDS) method introduced by Beik-Mohammadi et al. (2024). Later, we explain how NCDS can be improved through the introduction of a new data-driven regularization technique and modifications to the symmetry of its Jacobian formulation. Our main goal is to design a flexible neural architecture that is guaranteed to always output a contractive vector field. First, we introduce contraction theory as it is prerequisite to our design.

2.1 *Background: contractive 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 C^1 function, and $\dot{\boldsymbol{x}}_t = {}^{\mathrm{d}\boldsymbol{x}}/_{\mathrm{d}t}$ denotes temporal differentiation. As depicted in Fig. 3, 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 and Slotine 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 and I. Fossen 2010; Tsukamoto et al. 2021).

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 $\delta\boldsymbol{x}^\mathsf{T}\delta\boldsymbol{x}$ is

$$\frac{\mathrm{d}}{\mathrm{d}t}(\delta\boldsymbol{x}^\mathsf{T}\delta\boldsymbol{x}) = 2\delta\boldsymbol{x}^\mathsf{T}\delta\dot{\boldsymbol{x}} = 2\delta\boldsymbol{x}^\mathsf{T}\boldsymbol{J}(\boldsymbol{x})\delta\boldsymbol{x}. \tag{1}$$

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

Definition 1. Contraction stability (Lohmiller and Slotine 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 $\delta\boldsymbol{x}^\mathsf{T}\delta\boldsymbol{x}$ 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,*

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

The above analysis can be generalized to account for a more general notion of distance of the form $\delta\boldsymbol{x}^\mathsf{T}\boldsymbol{M}(\boldsymbol{x})\delta\boldsymbol{x}$, where

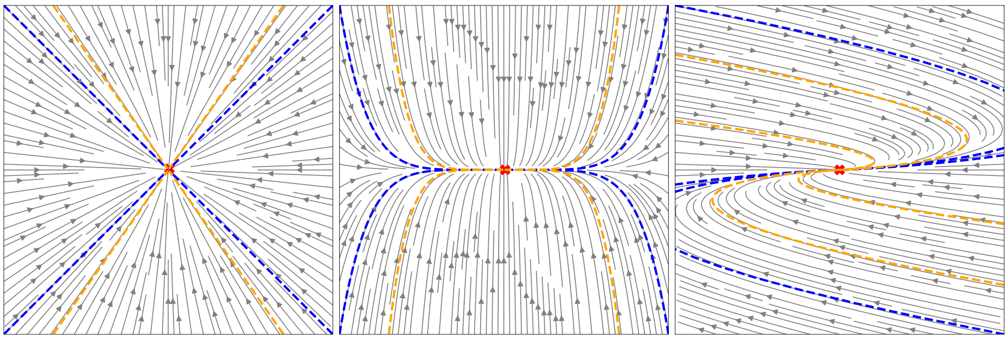


Figure 2. The effect of eigenvalue differences and asymmetry of the Jacobian on the contraction behavior of the vector field. *Left*: In a contractive dynamical system with a symmetric Jacobian, when the eigenvalues are equal, the system contracts uniformly in all directions. *Middle*: In a contractive 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.

$\boldsymbol{M}(\boldsymbol{x})$ is a positive-definite matrix known as the Riemannian contraction metric (Tsukamoto et al. 2021; Dawson et al. 2023). In our work, we learn a dynamical system f so that it is inherently contractive since its Jacobian $\boldsymbol{J}(\boldsymbol{x})$ fulfills the condition given in equation 2. Consequently, it is not necessary to separately learn a contraction metric as an identity matrix suffices.

2.2 *Neural contractive dynamical systems*

From Definition 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,

$$\hat{\boldsymbol{J}}_f(\boldsymbol{x}) = -(\boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{x})^\mathsf{T} \boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{x}) + \epsilon\, \mathbb{I}_D), \tag{3}$$

where $\boldsymbol{J}_{\boldsymbol{\theta}} : \mathbb{R}^D \to \mathbb{R}^{D\times D}$ is a neural network parameterized by $\boldsymbol{\theta}$, $\epsilon \in \mathbb{R}^+$ 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 $-\epsilon$.

Next, we take inspiration from Lorraine and Hossain (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,

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}) = \dot{\boldsymbol{x}}_0 + \int_0^1 \hat{\boldsymbol{J}}_f\left(c\left(\boldsymbol{x}, t, \boldsymbol{x}_0\right)\right) \dot{c}(\boldsymbol{x}, t, \boldsymbol{x}_0)\mathrm{d}t, \tag{4}$$

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

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 equation 4 enables accurate reconstruction of the velocities $\dot{\boldsymbol{x}}_i$ given the state $\boldsymbol{x}_i$. This is achieved through the velocity reconstruction loss,

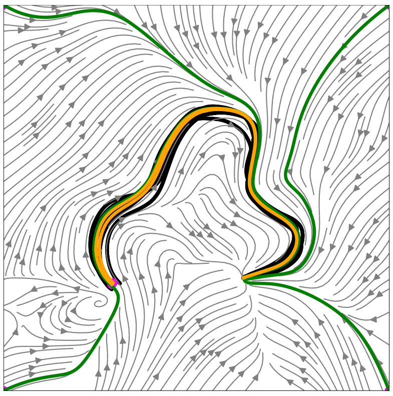


Figure 3. The learned vector field (grey) and demonstrations (black). Yellow and green trajectories show path integrals starting from demonstration starting points and random points, respectively.

$$\mathcal{L}_{\text{vel}} = \frac{1}{N}\sum_{i=1}^{N} \left\| \dot{\boldsymbol{x}}_i - \hat{\dot{\boldsymbol{x}}}_i \right\|^2, \tag{5}$$

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. 10.

Note that the integral in equation 4 resembles the neural ordinary differential equations (Chen et al. 2018b), 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 $\hat{\boldsymbol{J}}_f(\boldsymbol{x})$.

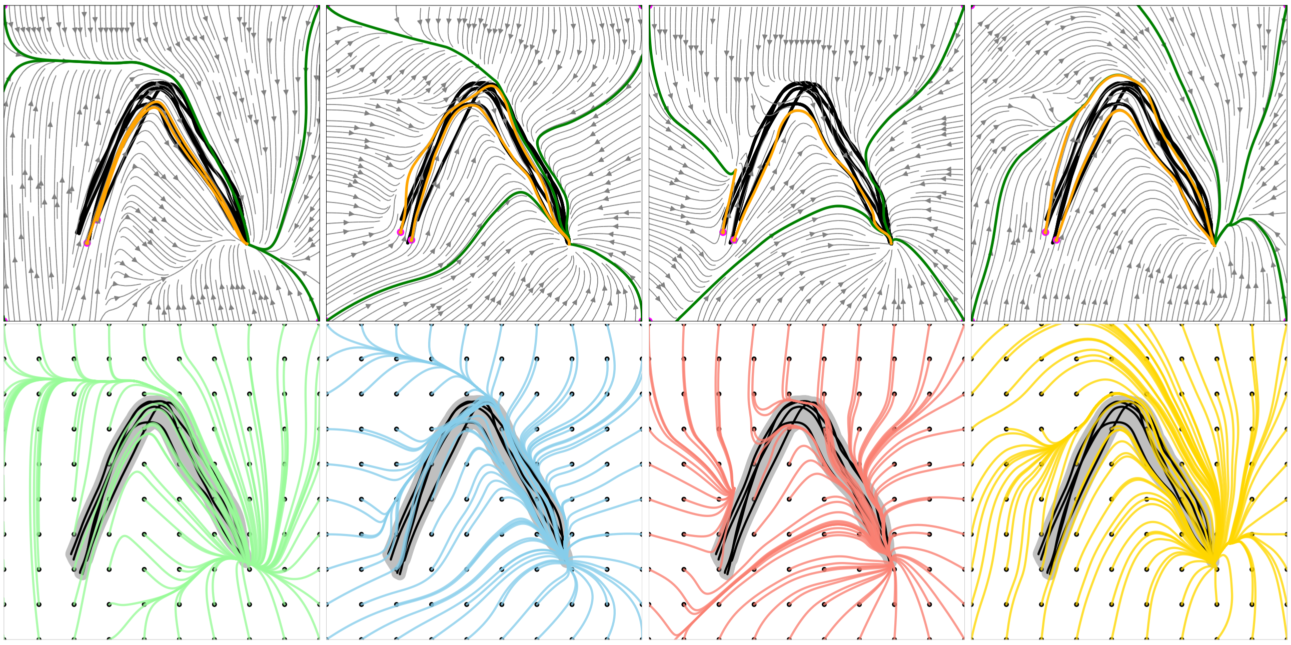


Figure 4. Comparison of 2D vector fields learned by using the following regularization approaches: *(a)*: constant regularization, *(b)*: state-independent regularization vector using basic optimization, *(c)*: state-dependent regularization vector utilizing neural network, and *(d)*: eigenvalue regularization.

We call this the *neural contractive dynamical system (NCDS)*, first introduced in our previous work (Beik-Mohammadi et al. 2024). This approach offers two key benefits: *(1)* it allows the use of any smooth neural network $\boldsymbol{J_\theta}$ as the base model, and *(2)* training can be performed with ordinary *unconstrained* optimization, unlike previous approaches.

Figure 3 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 equation 3 is characterized by two main components: the symmetric matrix $\boldsymbol{J_\theta}(\boldsymbol{x})^\mathsf{T}\boldsymbol{J_\theta}(\boldsymbol{x})$, and the regularization term $\epsilon\, \mathbb{I}_D$. 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.

Although NCDS (Beik-Mohammadi et al. 2024) demonstrated significant potential in learning complex contractive dynamical systems, there remains an opportunity to further explore the individual impact of each component. In the following two sections, we will explore the effects of these elements and propose improvements to enhance their performance. From this point forward, we will refer to the NCDS formulated with equations 3 and 4 as *vanilla NCDS*.

2.2.1 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 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. In vanilla NCDS, the regularization method, here referred to as *constant regularization*, adds a small constant to the diagonal terms of the Jacobian $\boldsymbol{J}_\theta^\mathsf{T}\boldsymbol{J}_\theta$. Therefore, the final learned eigenvalues are influenced by both the neural network parameters and the applied regularization. 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 can only represent a fixed point in space rather than a trajectory (Fig. 2–*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. 2–*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. 2–*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. 2–*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.

Given this analysis, we propose to better control the system's contraction through regularization by considering the following three approaches: *(1) state-independent regularization vector*, *(2) state-dependent regularization vector*, and *(3) eigenvalue regularization*. We also propose a metric to assess the system's contraction properties relative to the demonstration region in all scenarios. This metric quantifies the duration, in time steps, that each path integral

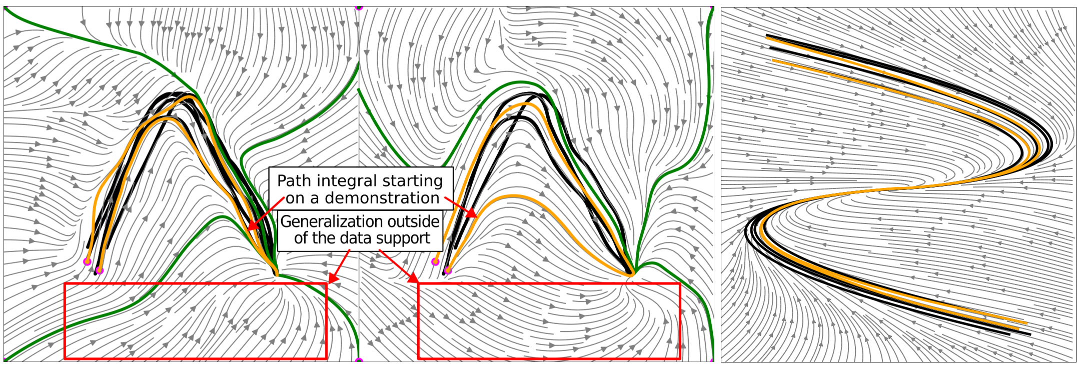


Figure 5. Comparison of 2D vector fields learned with different Jacobian formulations. *Left:* Vector field obtained using the symmetric Jacobian formulation. *Middle:* Vector field obtained using the asymmetric Jacobian formulation. The learned vector field (grey) and demonstrations (black). Yellow 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.

remained within this region. A higher count of time steps inside the region indicates that the trajectories converge faster, thus suggesting a stronger contraction. We now proceed to explain these different regularization approaches for NCDS.

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

$$\hat{\boldsymbol{J}}_f(\boldsymbol{x}) = -(\boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{x})^\mathsf{T} \boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{x}) + \mathrm{diag}(\boldsymbol{\epsilon})). \tag{6}$$

We propose two distinct methods to learn the vector $\boldsymbol{\epsilon}$.

State-independent: In this method, $\boldsymbol{\epsilon}$ is assumed to be independent of the system's state $\boldsymbol{x}$. The vector $\boldsymbol{\epsilon}$ is learned by minimizing the following loss function,

$$\mathcal{L}_{\boldsymbol{\epsilon}} = -\beta \sum_{n=2}^{D} |\epsilon_1 - \epsilon_n|^2 , \tag{7}$$

where β is a weight, and $\epsilon_1, \ldots, \epsilon_D$ are elements of the regularization vector $\boldsymbol{\epsilon}$. Thus, the overall loss is defined as $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{vel}} + \mathcal{L}_{\boldsymbol{\epsilon}}$. Figure 4–*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. 4–*a*. This is quantitatively supported by the contraction measure statistics reported in Fig. 7, showing that the trajectories converge approximately 276% faster towards the data support.

State-dependent: In this approach, the regularization vector $\boldsymbol{\epsilon} = g(\boldsymbol{x})$ is computed as a function of the state $\boldsymbol{x}$ using a neural network $g_{\boldsymbol{\delta}}(\boldsymbol{x}) : \mathbb{R}^D \to \mathbb{R}^D$ with parameters $\boldsymbol{\delta}$. The parameters $\boldsymbol{\delta}$ are learned via back propagation, which aims at minimizing the loss introduced in equation 7. Figure 4–*c* shows a contractive dynamical system using this state-dependent method, exhibiting faster trajectory convergence compared to the naive constant regularization (Fig. 4–*a*). Figure 7 shows that trajectories converge approximately 100% faster towards the data support. However, we noticed that the neural networks $g_{\boldsymbol{\delta}}$ might produce very small regularization vectors, potentially generating spurious attractors.

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, $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_D)$. To incorporate regularization directly on the eigenvalues, we can use the same loss as in equation 7, 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,

$$\mathcal{L}_{\boldsymbol{\epsilon}} = -\beta \sum_{n=1}^{D} |\lambda_i - \lambda_n|^2 , \tag{8}$$

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 4–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. 4–a. This is evident from the contraction measure statistics reported in Fig. 7, 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. 4 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.

Next, we turn our attention to the other key component of equation 3, namely the symmetric matrix $\boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{x})^\mathsf{T} \boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{x})$. 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.

2.2.2 Asymmetry of the Jacobian : Vanilla NCDS builds on a symmetric Jacobian, as formulated in equation 3. 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:

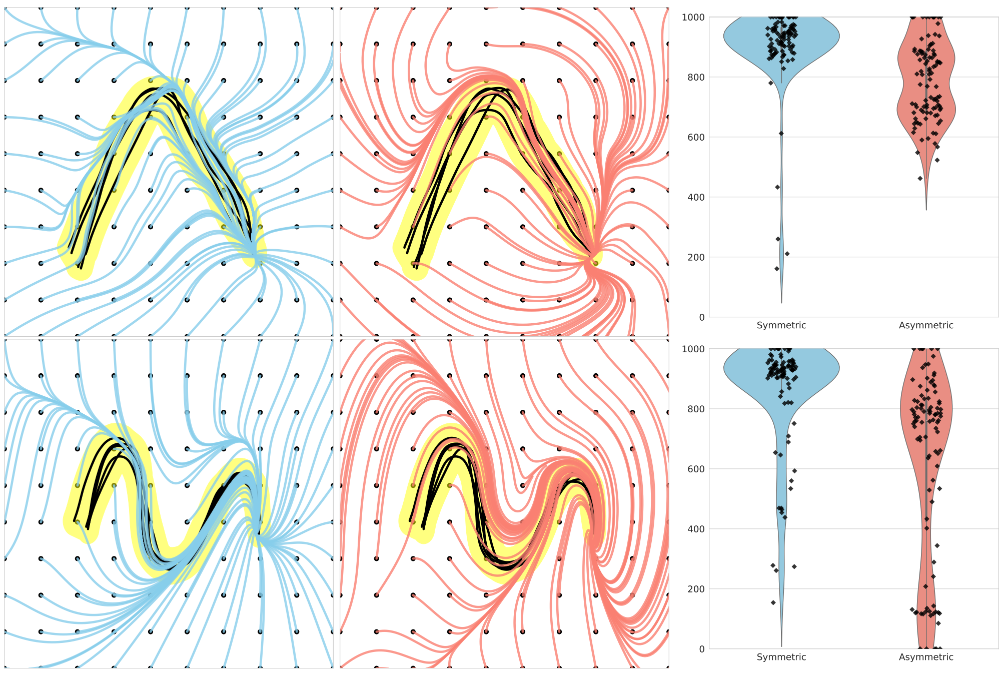


Figure 6. Comparison between asymmetric and symmetric Jacobians in learning contractive dynamical systems. *Left:* Generalization behavior of a system with a symmetric Jacobian, visualized using path integrals originating from a 10×10 equidistant grid. The yellow region represents the demonstration area, 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.

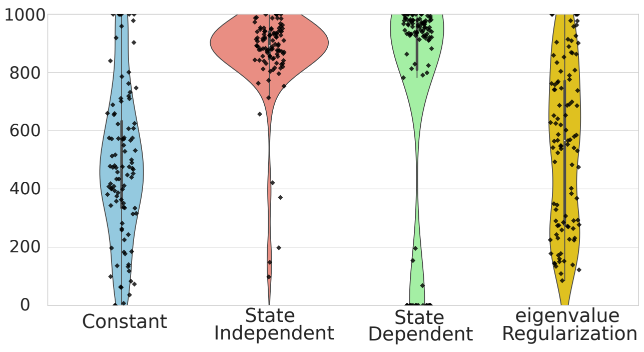


Figure 7. The distribution of the duration, in time steps, that each path integral remained within the demonstration region across four different regularization methods: *Constant*, *State Independent*, *State Dependent*, and *Eigenvalue Regularization*. A higher count of time steps inside the region indicates that trajectories likely converge faster and exhibit stronger contraction properties. Individual data points are represented by black dots.

$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x}$ with $\boldsymbol{A} = \begin{pmatrix} -1 & 4 \\ 0 & -1 \end{pmatrix}$. The vector field produced by this system is shown in Fig. 2–*right*.

Therefore, we can enhance the NCDS flexibility by reformulating the system's Jacobian to incorporate both a symmetric and a skew-symmetric component. This can be achieved by parameterizing the Jacobian using two separate neural networks: one generating the symmetric Jacobian, denoted as $\boldsymbol{J}_{\boldsymbol{\theta}}$, and the other generating the skew-symmetric Jacobian, denoted as $\boldsymbol{J}_{\boldsymbol{\phi}}$. The combined Jacobian is then

$$\hat{\boldsymbol{J}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) = \hat{\boldsymbol{J}}_{\boldsymbol{\theta}}(\boldsymbol{x}) + \hat{\boldsymbol{J}}_{\boldsymbol{\phi}}(\boldsymbol{x}), \tag{9}$$

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

$$\hat{\boldsymbol{J}}_{\boldsymbol{\phi}}(\boldsymbol{x}) = \begin{bmatrix} 0 & -\boldsymbol{J}_{\boldsymbol{\phi},0}(\boldsymbol{x}) & \boldsymbol{J}_{\boldsymbol{\phi},1}(\boldsymbol{x}) \\ \boldsymbol{J}_{\boldsymbol{\phi},0}(\boldsymbol{x}) & 0 & -\boldsymbol{J}_{\boldsymbol{\phi},2}(\boldsymbol{x}) \\ -\boldsymbol{J}_{\boldsymbol{\phi},1}(\boldsymbol{x}) & \boldsymbol{J}_{\boldsymbol{\phi},2}(\boldsymbol{x}) & 0 \end{bmatrix}. \tag{10}$$

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. 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–*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 equation 9. This is compared to the learned contractive vector field shown in Fig. 5–*left*, which has only a symmetric Jacobian, as described in equation 3. 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–*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–*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–*middle*), the system initially diverges the path integrals before redirecting them back to the target. Conversely, the system with the symmetric Jacobian (Fig. 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

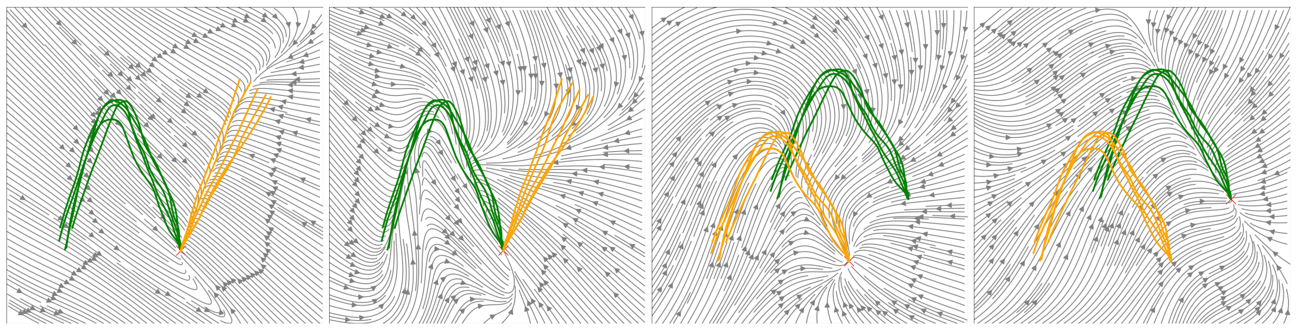


Figure 8. CNCDS on 2D trajectories from the LASA handwriting dataset. Plots (a) and (b) show CNCDS conditioned on the trajectory shape. Plots (c) and (d) display conditioning on the trajectory target, where the conditional value is chosen to be 0 and 1, respectively.

multiple path integrals starting from an equidistant grid around the demonstration trajectories. As shown in Fig. 6–*left* and 6–*middle*, the yellow region defines the area where the demonstrations, depicted as black curves, reside. Figures 6–*left* and 6–*middle* display all the path integrals generated by NCDS with symmetric and asymmetric Jacobians, respectively. Figure 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. It should be noted that both results in Fig. 5 and Fig. 6 are obtained using the *state-independent regularization vector*, with the only distinction being the symmetry properties of the Jacobian matrix.

We argue that the superior performance of NCDS with a symmetric Jacobian is due to its ability to locally approximate the behavior of dynamical systems that have a non-symmetric Jacobian. This is shown in Fig. 5–*right*, where NCDS with a symmetric Jacobian successfully learns the dynamics of an asymmetric system, shown in Fig. 2–*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 and Fig. 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. 4.1) and the robotic and human motion results (Sec. 4.3), have not identified any dynamical systems where the symmetric Jacobian approach failed to effectively capture and learn the underlying dynamics.

Having completed our discussion on improving the Jacobian formulation in equation 3, 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.

2.3 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 and Chung 2021b)—or, in other cases, generating a contraction metric for every condition when using Neural Contraction Metrics (NCMs) (Tsukamoto and Chung 2021a). 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 vanilla NCDS with the ability of learning multimodal tasks, which depend on context variables such as variable targets, using a single NCDS module.

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 equation 3 as,

$$\hat{\boldsymbol{J}}_f([\boldsymbol{x},\boldsymbol{\varpi}]) = -(\boldsymbol{J}_{\boldsymbol{\theta}}([\boldsymbol{x},\boldsymbol{\varpi}])^{\mathsf{T}}\boldsymbol{J}_{\boldsymbol{\theta}}([\boldsymbol{x},\boldsymbol{\varpi}])+\operatorname{diag}(\boldsymbol{\epsilon})). \quad (11)$$

Figure 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 8–a and Fig. 8–b display the vector fields resulting from conditioning on the trajectory shape. Figure 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. 8–b displays the vector field for the *line* motion. Furthermore, Fig. 8–c and Fig. 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. 4.4, 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. 10. In the next section, we will focus on how to equip NCDS with obstacle avoidance capabilities using modulation matrices.

2.4 Obstacle avoidance via matrix modulation

In this section, we review the matrix modulation technique employed by vanilla NCDS (Beik-Mohammadi et al. 2024) for obstacle avoidance. Subsequently, in Sec. 3.4, 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.

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 $\dot{\boldsymbol{x}} = \boldsymbol{G}(\boldsymbol{x})f_{\boldsymbol{\theta}}(\boldsymbol{x})$ is both contractive and steers around the obstacle. Formally, given the modulation matrix $\boldsymbol{G}$, we can reshape the vector field

$$\hat{\dot{\boldsymbol{x}}} = \boldsymbol{G}(\boldsymbol{x})f_{\boldsymbol{\theta}}(\boldsymbol{x}), \tag{12}$$

$$\boldsymbol{G}(\boldsymbol{x}) = \boldsymbol{E}(\boldsymbol{x})\boldsymbol{D}(\boldsymbol{x})\boldsymbol{E}(\boldsymbol{x})^{-1}, \tag{13}$$

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

$$\boldsymbol{E}(\boldsymbol{x}) = [\boldsymbol{n}(\boldsymbol{x})\; \mathrm{e}_1(\boldsymbol{x}) \ldots \mathrm{e}_{d-1}(\boldsymbol{x})], \tag{14}$$

$$\boldsymbol{D}(\boldsymbol{x}) = \mathrm{diag}(\lambda_n(\boldsymbol{x})\lambda_\tau(\boldsymbol{x}), \ldots, \lambda_\tau(\boldsymbol{x})), \tag{15}$$

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 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 $\mathrm{e}(\boldsymbol{x})$. 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 $\hat{\dot{\boldsymbol{x}}} = \boldsymbol{G}(\boldsymbol{x})f_{\boldsymbol{\theta}}(\boldsymbol{x})$ still guarantees contractive stability, which can be proved by following the same proof provided by Huber et al. (2019). Figure 9 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, vanilla NCDS successfully generated safe trajectories by effectively avoiding the obstacle and ultimately reaching the target.

Having introduced all the components of NCDS, we are now prepared to explore how these concepts can be extended by using a latent variable model to learn vector fields within a low-dimensional latent space.

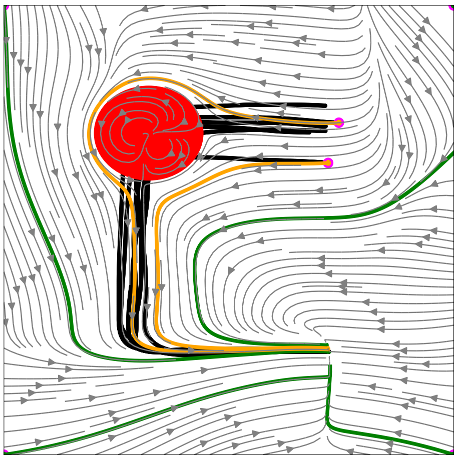


Figure 9. The obstacle locally reshapes the learned vector field using the modulation matrix. The gray contours represent the learned vector field, black trajectories depict the demonstrations, and orange/green trajectories are path integrals starting from both initial points of the demonstrations and plot corners. Magenta and red circles indicate the initial points of the path integrals and the obstacle, correspondingly.

3 Learning latent contractive dynamics

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. 2018a; 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.

3.1 Background: deep generative models and Riemannian manifolds

To obtain a low-dimensional representation of the demonstration data, the vanilla 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. 3.4 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

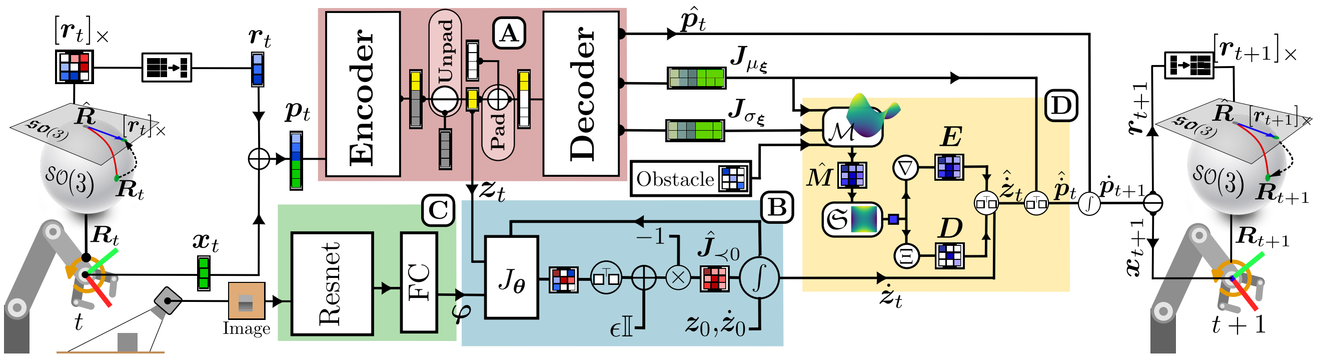


Figure 10. Architecture overview: a single iteration of NCDS simultaneously generating position and orientation dynamics. **(A) VAE (pink box):** The encoder processes the concatenated position-orientation data p_t, yielding a resulting vector that is subsequently divided into two components: the latent code z (yellow squares) and the surplus (gray squares). The Unpad function in equation 29 removes the unused segment. The unpadded latent code 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 equation 2. The numerical integral solver then computes the latent velocity $\dot{z}$. Later, using equation 30, $\dot{z}$ is mapped to the input-space velocity via the decoder's Jacobian J_{μ_ξ}. **(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 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 $G_{\mathcal{M}}(x)$.

space through pullback metrics. Therefore, we also introduce the necessary background on Riemannian manifolds.

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

$$p(z) = \mathcal{N}\left(z|\mathbf{0}, \mathbb{I}_d\right), \qquad z \in \mathcal{Z}; \tag{16}$$

$$p_{\varrho}(x|z) = \mathcal{N}\left(z|\mu_{\varrho}(z), \mathbb{I}_D\sigma^2_{\varrho}(z)\right), \qquad x \in \mathcal{X}. \tag{17}$$

where $\mu_\varrho : \mathcal{Z} \to \mathcal{X}$ and $\sigma_\varrho : \mathcal{Z} \to \mathbb{R}^D_+$ are deep neural networks with parameters ϱ estimating the mean and the variance of the posterior distribution $p_\varrho(x|z)$, 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,

$$\begin{aligned}\mathcal{L}_{ELBO} = {} & \mathbb{E}_{q_\xi(z|x)}\left[\log(p_\varrho(x|z))\right] \\ & - \mathrm{KL}\left(q_\xi(z|x)||p(z)\right),\end{aligned} \tag{18}$$

where $q_\xi(z|x) = \mathcal{N}(x|\mu_\xi(x), \mathbb{I}_d\sigma^2_\xi(x))$ approximates the posterior distribution $p(z|x)$ by two deep neural networks $\mu_\xi(x) : \mathcal{X} \to \mathcal{Z}$ and $\sigma_\xi(x)) : \mathcal{X} \to \mathbb{R}^d_+$. The posterior distribution $p_\xi(z|x)$ is called *inference* or *encoder* distribution, while the generative distribution $p_\varrho(x|z)$ is known as the *generator* or *decoder*. Later, we employ a VAE to capture a low-dimensional latent representation of the training data, which not only allows us to learn complex high-dimensional skills in a low-dimensional space but also enables the learning of Riemannian manifolds.

Injective flows: 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 and Turner 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 and Cranmer (2020) proposed an *injective flow*, which implements a zero-padding operation (see Sec. 3.2 and equation 28) on the latent variables alongside a diffeomorphic decoder, such that the resulting function is injective. Later, using the diffeomorphic properties of these models, we decode the learned contractive vector field from the latent space via the decoder of the injective flows.

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). 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 $\mathbb{R}^d$. In this paper, we use the mapping function Ω to represent a manifold $\mathcal{M}$ immersed in the ambient space $\mathcal{X}$ defined as,

$$\mathcal{M} = \Omega(\mathcal{Z}) \quad \text{with} \quad \Omega : \mathcal{Z} \to \mathcal{X}, \tag{19}$$

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

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

$$\mathcal{L}_\zeta = \int_0^1 \|\partial_t \Omega(\zeta(t))\| \mathrm{d}t. \tag{20}$$

This length can be reformulated using the chain rule as,

$$\mathcal{L}_\zeta = \int_0^1 \sqrt{\dot{\zeta}(t)^\mathsf{T} M(\zeta(t))\dot{\zeta}(t)}\mathrm{d}t, \tag{21}$$

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

$$\boldsymbol{M}(\boldsymbol{z}) = \boldsymbol{J}_{\Omega}(\boldsymbol{z})^{\mathsf{T}}\boldsymbol{J}_{\Omega}(\boldsymbol{z}). \quad (22)$$

Here, $\boldsymbol{J}_{\Omega}(\boldsymbol{z})$ is the Jacobian of the mapping function Ω. This metric can be used to measure local distances in $\mathcal{Z}$. The shortest path on the manifold, also known as the geodesic, can be computed given the curve length in equation 21.

Learning Riemannian manifolds with VAEs: We here explain the link between VAEs with an injective decoder and Riemannian geometry. It should be mentioned that learning Riemannian manifolds is independent of the injectivity of the decoder. However, the injection is crucial in transferring stability guarantees from the latent space to the ambient space. To begin, we define the VAE generative process of equation 17 as a stochastic function,

$$f_{\boldsymbol{\varrho}}(\boldsymbol{z}) = \mu_{\boldsymbol{\varrho}}(\boldsymbol{z}) + \mathrm{diag}(\varepsilon)\sigma_{\boldsymbol{\varrho}}(\boldsymbol{z}), \quad \varepsilon \sim \mathcal{N}(\boldsymbol{0}, \mathbb{I}_D), \quad (23)$$

where $\mu_{\boldsymbol{\varrho}}(\boldsymbol{z})$ and $\sigma_{\boldsymbol{\varrho}}(\boldsymbol{z})$ are decoder mean and variance neural networks, respectively. Also, $\mathrm{diag}(\cdot)$ 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 and Welling 2014), which can be interpreted as samples generated out of a random projection of a manifold jointly spanned by $\mu_{\boldsymbol{\varrho}}$ and $\sigma_{\boldsymbol{\varrho}}$ (Eklund and Hauberg 2019).

Riemannian manifolds may arise from mapping functions between two spaces as in equation 19. As a result, equation 23 may be seen as a stochastic version of the mapping function of equation 19, which in turn defines a Riemannian manifold (Hauberg 2019). We can now write the stochastic form of the Riemannian metric of equation 22. To do so, we first recast the stochastic function equation 23 as follows (Eklund and Hauberg 2019),

$$f_{\boldsymbol{\phi}}(\boldsymbol{z}) = \begin{pmatrix}\mathbb{I}_D, & \mathrm{diag}(\varepsilon)\end{pmatrix} \begin{pmatrix}\mu_{\boldsymbol{\varrho}}(\boldsymbol{z}) \\ \sigma_{\boldsymbol{\varrho}}(\boldsymbol{z})\end{pmatrix} = \boldsymbol{P}\, s(\boldsymbol{z}), \quad (24)$$

where $\boldsymbol{P}$ is a random matrix, and $s(\boldsymbol{z})$ is the concatenation of $\mu_{\boldsymbol{\varrho}}(\boldsymbol{z})$ and $\sigma_{\boldsymbol{\varrho}}(\boldsymbol{z})$. 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 $\mu_{\boldsymbol{\varrho}}(\boldsymbol{z})$ and variance $\sigma_{\boldsymbol{\varrho}}(\boldsymbol{z})$, the metric is likewise based on a mixture of both as follows,

$$\boldsymbol{M}(\boldsymbol{z}) = \boldsymbol{J}_{\mu_{\boldsymbol{\varrho}}}(\boldsymbol{z})^{\mathsf{T}}\boldsymbol{J}_{\mu_{\boldsymbol{\varrho}}}(\boldsymbol{z}) + \boldsymbol{J}_{\sigma_{\boldsymbol{\varrho}}}(\boldsymbol{z})^{\mathsf{T}}\boldsymbol{J}_{\sigma_{\boldsymbol{\varrho}}}(\boldsymbol{z}), \quad (25)$$

where $\boldsymbol{J}_{\mu_{\boldsymbol{\varrho}}}(\boldsymbol{z})$, $\boldsymbol{J}_{\sigma_{\boldsymbol{\varrho}}}(\boldsymbol{z})$, $\boldsymbol{J}_{\mu_{\boldsymbol{\varrho}}}(\boldsymbol{z})^{\mathsf{T}}$, and $\boldsymbol{J}_{\sigma_{\boldsymbol{\varrho}}}(\boldsymbol{z})^{\mathsf{T}}$ are respectively the Jacobian of $\mu_{\boldsymbol{\varrho}}(\boldsymbol{z})$ and $\sigma_{\boldsymbol{\varrho}}(\boldsymbol{z})$ and their corresponding transpose evaluated at $\boldsymbol{z} \in \mathcal{Z}$, with $\mathcal{Z}$ being the VAE low-dimensional latent space. With access to this metric, we can reshape it to adjust the curvature of the manifold in specific regions. This reshaping allows us to locally increase the curvature—and therefore the metric volume—on parts of the manifold where obstacles are located. We later discuss how this adjusted curvature can be transformed into a metric that aids in obstacle avoidance within the latent space. In this study, we employ a reshaped ambient metric to integrate obstacle information (Beik-Mohammadi et al. 2023). Specifically, assuming that an obstacle is modeled using a Gaussian-like function, we define the ambient metric as:

$$\boldsymbol{M}_{\mathcal{X}}(\boldsymbol{x}) = \left(1 + \mathfrak{w} \exp\left(\frac{-\|\boldsymbol{x}-\boldsymbol{o}\|^2}{2r^2}\right)\right)\mathbb{I}_3, \quad \boldsymbol{x} \in \mathbb{R}^3, \quad (26)$$

where $\mathfrak{w} > 0$ scales the obstacle cost, and $\boldsymbol{o} \in \mathbb{R}^3$, $r > 0$ represent the obstacle's position and radius, respectively. The orientation metric is assumed to be identity. Note that we use a Gaussian-like function to represent the obstacle here; however, more complex representations, such as point cloud-based meshes, can also be utilized. Then, the corresponding Riemannian metric in the latent space is then given by,

$$\boldsymbol{M}(\boldsymbol{z}) = \boldsymbol{J}_{\mu_{\boldsymbol{\varrho}}}(\boldsymbol{z})^{\mathsf{T}}\boldsymbol{M}_{\mathcal{X}}\boldsymbol{J}_{\mu_{\boldsymbol{\varrho}}}(\boldsymbol{z}) + \boldsymbol{J}_{\sigma_{\boldsymbol{\varrho}}}(\boldsymbol{z})^{\mathsf{T}}\boldsymbol{M}_{\mathcal{X}}\boldsymbol{J}_{\sigma_{\boldsymbol{\varrho}}}(\boldsymbol{z}). \quad (27)$$

Note that changing the obstacle positions does not require re-training the VAE, as this only changes the ambient metric $\boldsymbol{M}_{\mathcal{X}}(\boldsymbol{x})$.

3.2 Latent NCDS

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 and Slotine 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 1. Contraction invariance under diffeomorphisms (Manchester and Slotine 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 1, we learn a VAE with an 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 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 and Cranmer (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,

$$\mu = \iota_K \circ \cdots \circ \iota_1 \circ \mathrm{Pad}, \quad (28)$$

where $\mathrm{Pad}(\boldsymbol{z}) = [z_1 \cdots z_d \; 0 \cdots 0]^{\mathsf{T}}$ represents a D-dimensional vector $\boldsymbol{z}$ with additional $D-d$ zeros. 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 $\mu_{\boldsymbol{\xi}}$ follows the architecture in equation 28. Empirically, we have found that training stabilizes when the variational encoder takes the form $q_{\boldsymbol{\xi}}(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z} \mid \mu_{\boldsymbol{\xi}}^{-1}(\boldsymbol{x}),\, \mathbb{I}_d\sigma_{\boldsymbol{\xi}}^2(\boldsymbol{x}))$, where $\mu_{\boldsymbol{\xi}}^{-1}$ is the approximate inverse of $\mu_{\boldsymbol{\xi}}$ given by,

$$\mu_{\boldsymbol{\xi}}^{\sim 1} = \text{Unpad} \circ \iota_1^{-1} \circ \cdots \circ \iota_K^{-1}, \tag{29}$$

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. 10.

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 $\mu_{\boldsymbol{\xi}}$, computed as $\boldsymbol{J}_{\mu_{\boldsymbol{\xi}}}(\boldsymbol{z}) = {}^{\partial \mu_{\boldsymbol{\xi}}}/_{\partial \boldsymbol{z}}$. This enables the decoding process formulated as below,

$$\dot{\boldsymbol{x}} = \boldsymbol{J}_{\mu_{\boldsymbol{\xi}}}(\boldsymbol{z})\dot{\boldsymbol{z}}. \tag{30}$$

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. 2.2). Then, the latent velocities* $\dot{\boldsymbol{z}}$ given by such a contractive dynamical system can be mapped to the data space $\mathcal{X}$ using equation 30. 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 $\boldsymbol{z}_0 = \mu^{\sim 1}(\boldsymbol{x}_0)$. Note that the movement from $\boldsymbol{x}_0$ to $f(\boldsymbol{z}_0)$ need not be contractive, but this is a finite-time motion, after which the system is contractive.

3.3 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 1.

3.3.1 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* $\mathcal{SO}(3) = \{\boldsymbol{R} \in \mathbb{R}^{3\times 3} \,|\, \boldsymbol{R}^\mathsf{T}\boldsymbol{R} = \mathbb{I}, \det(\boldsymbol{R}) = 1\}$. Every Lie group is associated with its Lie algebra, which represents the tangent space at its origin (see Fig. 11–*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 $\boldsymbol{r} = [r_x, r_y, r_z] \in \mathbb{R}^3$.

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,

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

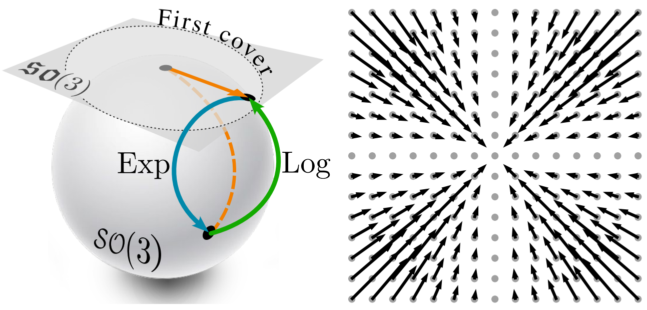


Figure 11. *Left*: Aspects of the Lie group $\mathcal{SO}(3)$. *Right*: An illustration of the function $b(\boldsymbol{x})$ in two dimensions.

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* $\mathcal{S}^3 = \{\boldsymbol{q} \in \mathcal{S}^3 \subset \mathbb{R}^4 \,|\, \|\boldsymbol{q}\| = 1\}$. 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,

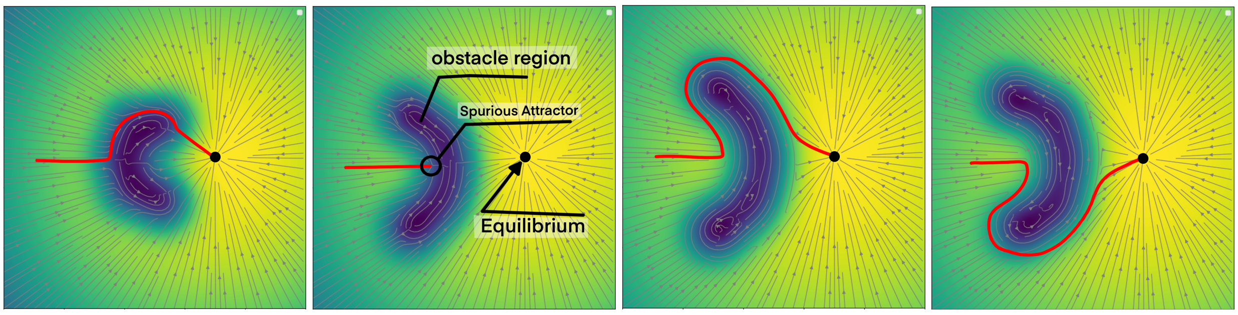
$$\text{Exp}(\boldsymbol{v}) = \cos\left(\frac{\|\boldsymbol{v}\|}{2}\right) + \sin\left(\frac{\|\boldsymbol{v}\|}{2}\right)\frac{\boldsymbol{v}}{\|\boldsymbol{v}\|},$$
$$\text{Log}(\boldsymbol{q}) = 2\arccos(q_0)\frac{(q_x, q_y, q_z)}{\sqrt{q_x^2 + q_y^2 + q_z^2}},$$

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.

3.3.2 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

*For training, the latent velocities are simply estimated by a numerical differentiation w.r.t the latent state $\boldsymbol{z}$.



(a) Convex obstacle without tangential vector field
(b) Concave obstacle without tangential vector field
(c) Concave obstacle with left tangential vector field
(d) Concave obstacle with right tangential vector field

Figure 12. Modulated contractive dynamical systems using a pullback metric derived from the decoder's Jacobian.

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 $\operatorname{Exp} \circ \mu$ in both cases. Unfortunately, even if μ is injective, we cannot ensure that $\operatorname{Exp} \circ \mu$ 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 $\operatorname{Exp} \circ \mu$ 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,

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

which smoothly (and invertibly) deforms the $[-1, 1]^D$ box into the unit ball (see Fig. 11–*right*). Then, the function,

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

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

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 10 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.

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

```
Input: Data: $\tau_n = \{\boldsymbol{x}_{n,t}, \boldsymbol{R}_{n,t}\}_{n=1}^N$,   $t \in [1, T_n]$
Output: Learned contractive dynamics parameters $\boldsymbol{\theta}$.
1  foreach trajectory $n$ do
2      foreach time step $t$ do
3          $\boldsymbol{r}_{n,t} = \operatorname{Log}(\boldsymbol{R}_{n,t})$   ▷ Extract skew-sym coeffs
4          $\boldsymbol{p}_{n,t} = [\boldsymbol{x}_{n,t}, \boldsymbol{r}_{n,t}]$   ▷ Form new state vector
5      end
6  end
7  $\boldsymbol{\xi}^* = \arg\min_{\boldsymbol{\xi}} \mathcal{L}_{\text{ELBO}}(\boldsymbol{p}_{n,t})$   ▷ Train the VAE
8  foreach trajectory $n$ do
9      foreach time step $t$ do
10         $\boldsymbol{z}_{n,t} = \mu_{\boldsymbol{\xi}^*}(\boldsymbol{p}_{n,t})$   ▷ Encode poses
11         $\dot{\boldsymbol{z}}_{n,t} = \frac{\boldsymbol{z}_{n,t+1} - \boldsymbol{z}_{n,t}}{\Delta t}$   ▷ Compute latent velocities
12     end
13 end
14 $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\text{Jac}}(\boldsymbol{p}_{n,t})$   ▷ Train Jacobian network
```

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
1  foreach time step $t$ do
2      $\boldsymbol{r}_t = \operatorname{Log}(\boldsymbol{R}_t)$   ▷ Extract skew-sym coeffs
3      $\boldsymbol{p}_t = [\boldsymbol{x}_t, \boldsymbol{r}_t]$   ▷ Form new state vector
4      $\boldsymbol{z}_t = \mu_{\boldsymbol{\xi}}(\boldsymbol{p}_t)$   ▷ Compute the latent state
5      $\hat{\dot{\boldsymbol{z}}}_t = f(\boldsymbol{z}_t)$   ▷ Compute the latent velocity
6      $\boldsymbol{J}_{\mu_{\boldsymbol{\xi}}}(\boldsymbol{z}_t) = \frac{\partial \mu_{\boldsymbol{\xi}}}{\partial \boldsymbol{z}_t}$   ▷ Compute the Jacobian of the
       decoder
7      $\dot{\boldsymbol{x}}_t = \boldsymbol{J}_{\mu_{\boldsymbol{\xi}}}(\boldsymbol{z}_t)\hat{\dot{\boldsymbol{z}}}_t$   ▷ Compute input space velocity
8  end
```

3.4 Latent obstacle avoidance and Riemannian safety regions

Section 3.1 described a Riemannian metric (equation 27) in the VAE latent space. Under this metric, shortest paths stay within the data support while avoiding obstacles. In this section, we build on this work and introduce a *modulation*

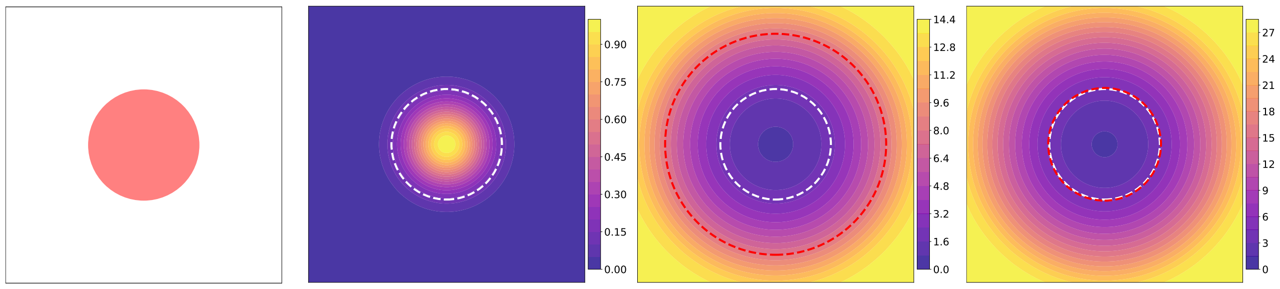


Figure 13. Illustration of the computation of the obstacle avoidance distance field for designing Riemannian safety regions. From left: (1) Obstacle representation (circle), (2) Metric volume around obstacle, (3) Metric inversion to compute distance map and problem with initial boundary misalignment (white vs. red circle for real vs. approximated boundary), (4) Rescaling with α to correct the boundary.

matrix, ensuring that our NCDS vector field also stays within data support and avoids obstacles.

We notice that the volume of the pullback Riemannian metric in equation 27 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 $\hat{f}$,

$$\hat{f}(\boldsymbol{z}) = \boldsymbol{G}_{\mathcal{M}}(\boldsymbol{z})f(\boldsymbol{z}). \tag{31}$$

To design this modulation matrix, we follow the recipe of Huber et al. (2022), described in Sec. 2.4, 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 equation 15. Specifically, we ensure that $\lambda_n(\boldsymbol{z}) : \mathbb{R}^d \to [0.0, 1.0]$ decreases towards 0.0 and $\lambda_\tau(\boldsymbol{z}) : \mathbb{R}^d \to [1.0, 2.0]$ increases towards 2.0 when the distance to the obstacle decreases. We give explicit expressions in Appendix 6.1.

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,

$$\boldsymbol{n}(\boldsymbol{z}) = \nabla\mathfrak{S}(\boldsymbol{z}) \quad \text{s.t.} \quad \boldsymbol{e}(\boldsymbol{z}) \perp \boldsymbol{n}(\boldsymbol{z}), \tag{32}$$

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 13 illustrates the computation of the distance field and Appendix 6.2 describes how we select the α factor.

The complete process from obstacle to distance field involves several key steps, as illustrated in Fig. 13. 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.

As Koptev (2023) discuss, the modulation in equation 13 can generate spurious attractors around concave regions of the obstacle's surface (Fig.12–*b*). Koptev (2023) suggests using a secondary tangential vector field that activates only when the velocity is zero to avoid this issue. Specifically,

$$\hat{f} = \boldsymbol{G}_{\mathcal{M}}(\boldsymbol{z}) \cdot f(\boldsymbol{z}) + \beta(\boldsymbol{x})\boldsymbol{G}_{\mathcal{M}}(\boldsymbol{z}) \cdot g(\boldsymbol{z}), \quad \beta \in [0, 1] \tag{33}$$

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.

4 Experiments

To evaluate the efficiency of 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. 4.6, 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. Demonstration videos are available at: https://sites.google.com/view/extendedncds/home.

Datasets. There are currently no established benchmarks for contraction-stable robot motion learning, so we focus on two datasets. First, we test our approach 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

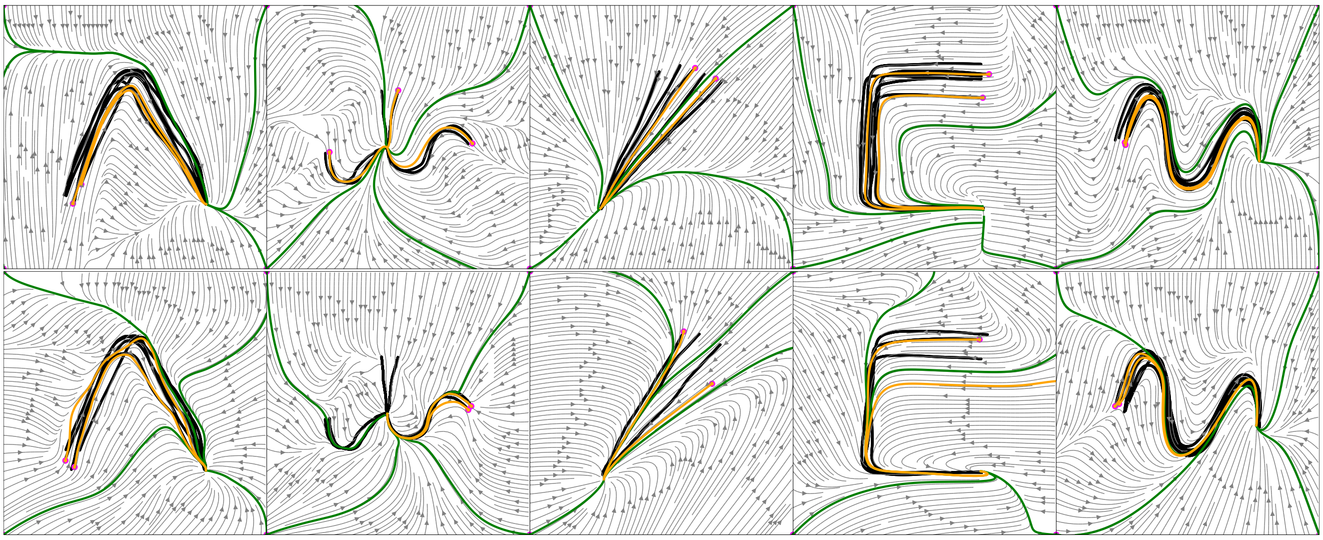


Figure 14. Visualization of the LASA-2D dataset: Gray contours represent the learned vector field, black trajectories depict demonstrations, and orange/green trajectories illustrate path integrals starting from the initial points of the demonstrations and plot corners. The magenta circles indicate the initial points of the path integrals.

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

Table 1. 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. 14.

state. Additionally, we omit the initial few points of each demonstration, to ensure that the only state exhibiting zero velocity is the target state. 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),

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

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.

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 2 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. 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 and Billard 2011). 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.

4.1 LASA dataset

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.

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 and Chung 2021a)	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 and Dani 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

Table 2. The present state-of-the-art literature pertaining to the learning of contractive dynamical systems.

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

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

Table 4. Average execution time (in milliseconds) of a single NCDS integration step for learning the vector field in different spaces.

Figure 14 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. 7). We observe that both NCDS approaches effectively capture and replicate the underlying dynamics. This is observed in the orange path integrals, starting from the initial points of the demonstrations, showing that both approaches can reproduce the demonstrated trajectories accurately.

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 1 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. 14. 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. 7). This is evidence that the contractive construction is a viable approach to controlling the extrapolation properties of the neural network.

Comparative Study on the LASA Dataset: Table 3 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. 16 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

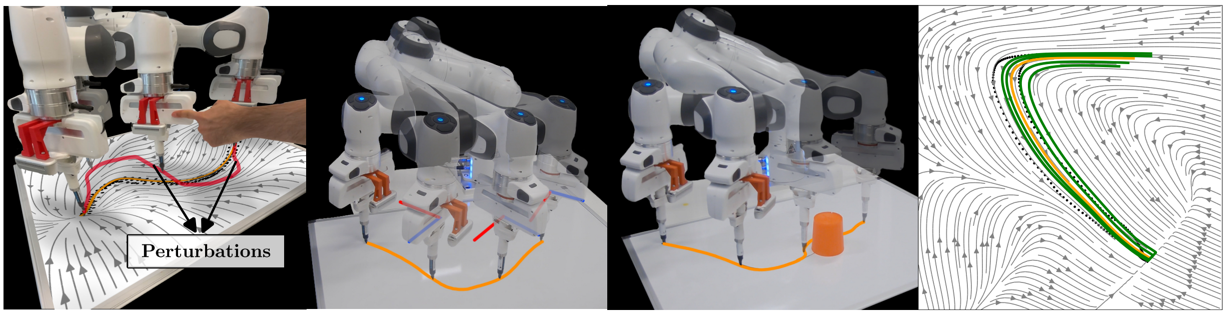


Figure 15. Robot experiments. The left two panels show robot experiments with orange and red paths illustrating unperturbed and perturbed motion. The first panel shows a learned vector field in $\mathbb{R}^3$ with a constant orientation. In the second panel, the vector field expands to $\mathbb{R}^3 \times \mathcal{SO}(3)$, 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.

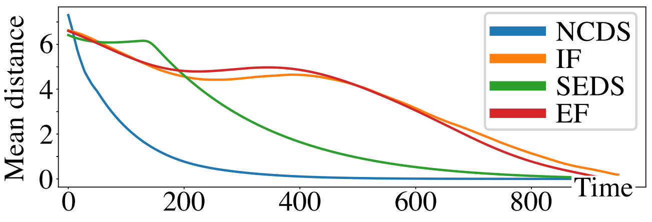


Figure 16. Average distance between random nearby trajectories over time for LASA-2D. Only NCDS monotonically decreases, i.e. it is the only contractive method.

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 3, 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. 17, 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.

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 3 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 4 compares 5 different experimental setups. Both the Variational Autoencoder (VAE) and Jacobian network $\hat{\boldsymbol{J}}_{\boldsymbol{\theta}}$, were implemented using PyTorch (Paszke et al. 2019). The VAE employs an injective generator based on $\mathcal{M}$-flows (Brehmer and Cranmer 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 4 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.

4.2 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. 2019). For the VAE, we used an injective generator based on $\mathcal{M}$-flows (Brehmer and Cranmer 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. 4.1. 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

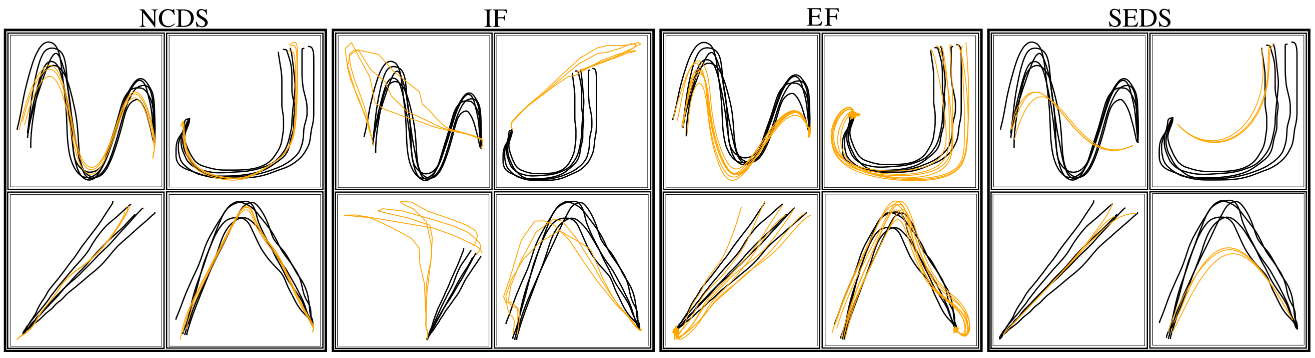


Figure 17. Path integrals generated using different methods on LASA-8D. Black curves are training data, while yellow are the learned path integrals. To construct the 8D dataset, we have concatenated 4 different 2D datasets.

the robot end-effector dynamics only evolve in $\mathbb{R}^3$. As Fig. 15-*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. 15. 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 15-*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 $\mathbb{R}^3 \times \mathcal{SO}(3)$.

4.2.1 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 equation 13. The third panel of Fig. 15 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.

4.3 *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. 4.1 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. 21–*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 able to learn high-dimensional dynamics. Figure 19–*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

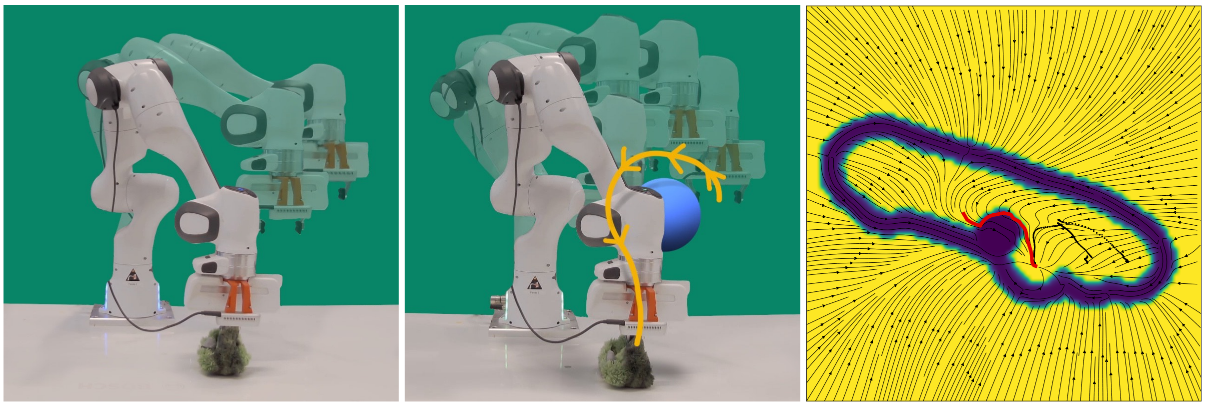


Figure 18. Obstacle avoidance via a Riemannian pullback metric. *Left*: The robot performs a grasping task without obstacle avoidance. *Middle*: The robot navigates around the obstacle by avoiding both the obstacle and the unsafe regions of the manifold. *Right*: The latent modulated vector field is locally reshaped around the obstacle and in areas outside the data support.

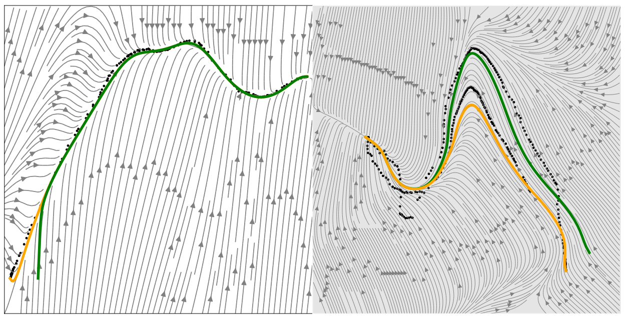


Figure 19. *Left:* Latent path integrals with NCDS trained in joint space $\mathbb{R}^{44}$. *Right:* Latent path integrals with NCDS trained on full human motion space: $\mathbb{R}^{44} \times \mathbb{R}^3 \times \mathcal{SO}(3)$. The background depicts the contours of the latent vector field, green and yellow curves depict the path integrals, and black dots represent demonstration in latent space.

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 19–*right* shows the latent path integrals generated by NCDS, while Fig. 21–*right* shows the corresponding motion. The upper row of the latter plot 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. 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.

4.4 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 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. 20 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.

4.5 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. 3.4, we use a Riemannian pullback metric computed via equation 27, to formulate the modulation matrix in equation 33. Specifically, we define the ambient metric $M_{\mathcal{X}}$ for the end-effector position as in equation 26.

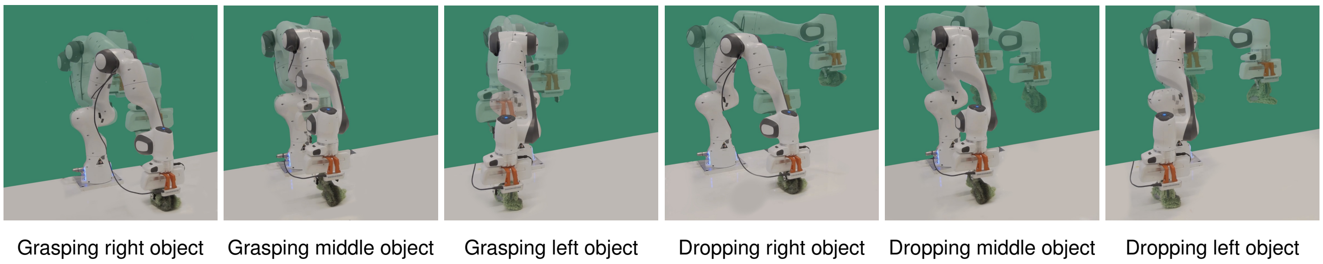


Figure 20. Grasping and dropping actions learned by the CNCDS for a soft object in three positions, conditioned on image input. The CNCDS is trained using cross-entropy loss, velocity reconstruction, and regularization with *ResNet18* image embeddings.

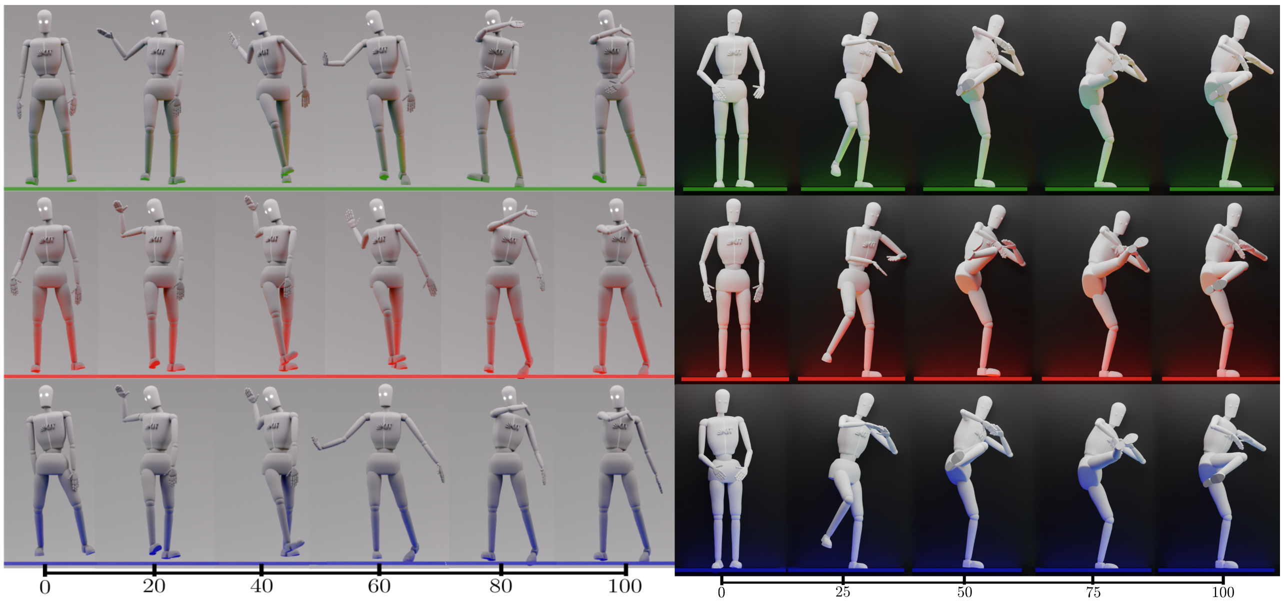


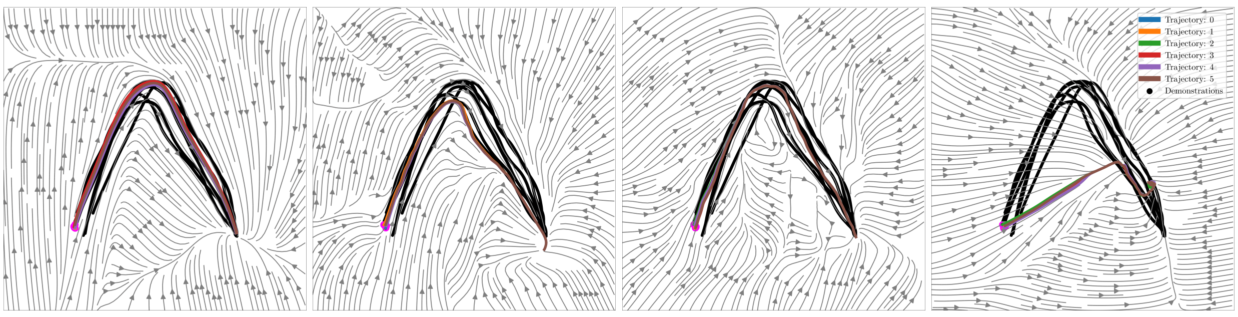
Figure 21. *Left*: Generated Human Motion with NCDS trained in joint space $\mathbb{R}^{44}$: The upper row depicts the progressive evolution of the demonstration motion from left to right. Meanwhile, the middle and bottom rows illustrate the motions generated by NCDS when the initial point is respectively 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.

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. 4.4, where vector field modulation is incorporated based on the Riemannian formulation introduced in Sec. 3.4. In Fig. 18, 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 results demonstrate that modulation based on the Riemannian metric allows the robot to effectively avoid obstacles while remaining within learned safety regions.

4.6 Ablation studies

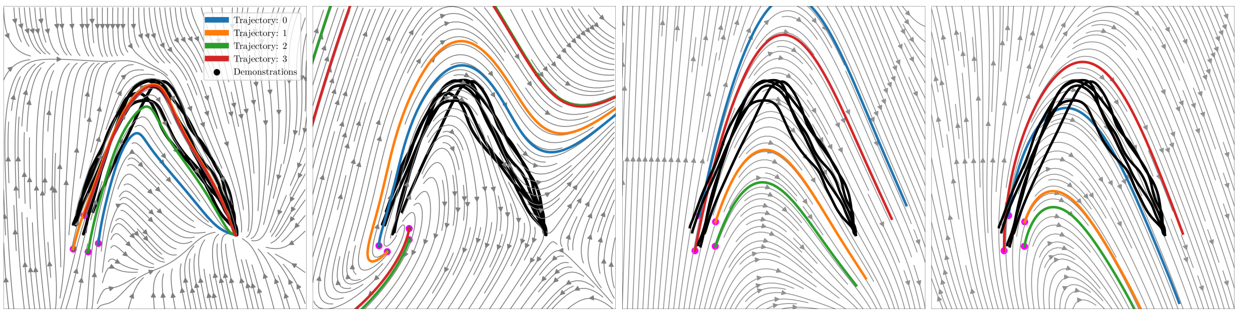
In this section, we perform ablation studies to analyze the impact of various components of our NCDS framework. By systematically altering or removing key elements, we aim to highlight the significance of the contraction constraint, the choice of network architecture, and the effect of different activation 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 $\hat{\boldsymbol{J}}_{\boldsymbol{\theta}}$. Figure 23 shows the path integrals generated by both contractive (Fig. 23-a) and unconstrained (Fig. 23-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



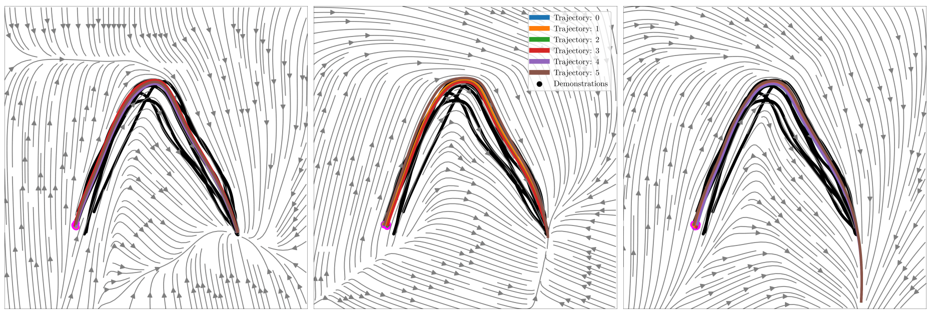
(a) $\epsilon = 10^{-4}$ **(b)** $\epsilon = 10^{-2}$ **(c)** $\epsilon = 10^{-1}$ **(d)** $\epsilon = 10$

Figure 22. Path integrals generated by NCDS trained using different regularization term ϵ.



(a) Contractive **(b)** Unconstrained **(c)** MLP **(d)** Neural ODE

Figure 23. 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.



(a) Tanh **(b)** Softplus **(c)** Sigmoid

Figure 24. Path integrals generated by NCDS trained using different activation functions.

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 $\dot{x} = f(x)$. Both networks are trained for 1000 epochs with the ADAM (Kingma and Ba 2014). Figure 23-c and 23-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. 24, both the Tanh and Softplus activation functions give superior performance, coupled with satisfactory 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.

5 Discussion and future work

In this paper, we proposed an enhanced framework for designing neural networks that are contractive, building upon the original Neural Contractive Dynamical Systems (NCDS, Beik-Mohammadi et al. (2024)). This allows us to create a flexible class of models capable of learning dynamical systems from demonstrations while retaining contractive stability guarantees, offering a more controlled generalization behavior. Key improvements include the introduction of Jacobian matrix regularization, where we explore both state-independent and state-dependent regularization strategies, as well as eigenvalue-based methods, to fine-tune the contraction properties and improve robustness and generalization.

Furthermore, we extended the NCDS framework by introducing asymmetry into the Jacobian matrix, incorporating both symmetric and skew-symmetric components. However, our evaluation showed that this modification did not provide any significant advantage over using only the symmetric part. This suggests that the NCDS with symmetric Jacobian is sufficient to locally approximate the behavior of dynamical systems with non-symmetric Jacobians, eliminating the need for the added complexity.

We also introduced the Conditional NCDS (CNCDS), which allows us to handle multiple motion skills by conditioning on task-related variables, such as target states, allowing a single NCDS module to adapt to multiple task conditions.

Finally, to enhance real-world applicability, we improve the model's obstacle avoidance capabilities by employing Riemannian safety regions on the NCDS latent space. Inspired by modulation matrix techniques, this approach reshapes the latent learned vector field near unsafe areas, enabling the system to avoid obstacles and out-of-data support regions, while maintaining contractive stability guarantees.

Overall, our extended NCDS framework significantly addresses the key limitations of the original model, introducing improvements in flexibility, robustness, and adaptability.

However, several limitations persist. The model remains sensitive to the choice of numerical integration schemes, particularly the reliance on adaptive step sizing to ensure reliable performance, which increases computation time and poses challenges for real-time applications. Moreover, the need to compute the Jacobian of the decoder at every step, in order to calculate the ambient velocity and Riemannian metric, further impacts performance, adding computational overhead. Furthermore, a limitation of implementing the conditional variable in CNCDS is its limited reactivity to changes in the conditioning variable. In other words, if the conditioning variable changes during skill execution, it is uncertain whether CNCDS will adapt seamlessly to these changes. We consider these issues minor compared to the many benefits that the NCDS framework provides.

Acknowledgements

This work was supported by a research grant (42062) from VILLUM FONDEN. This work was partly funded by the Novo Nordisk Foundation through the Center for Basic Research in Life Science (NNF20OC0062606). This project received funding from the European Research Council (ERC) under the European Union's Horizon Programme (grant agreement 101125003).

References

Beik-Mohammadi H, Hauberg S, Arvanitidis G, Figueroa N, Neumann G and Rozo L (2024) Neural contractive dynamical systems. In: *The Twelfth International Conference on Learning Representations*. URL https://openreview.net/forum?id=iAYIRHOYy8.

Beik-Mohammadi H, Hauberg S, Arvanitidis G, Neumann G and Rozo L (2021) Learning Riemannian manifolds for geodesic motion skills. In: *Robotics: Science and Systems (R:SS)*. URL https://roboticsconference.org/2021/program/papers/082/index.html.

Beik-Mohammadi H, Hauberg S, Arvanitidis G, Neumann G and Rozo L (2023) Reactive motion generation on learned riemannian manifolds. *The International Journal of Robotics Research (IJRR)* 42(10): 729–754. URL https://doi.org/10.1177/02783649231193046.

Billard A, Calinon S and Dillmann R (2016) Learning from humans. In: Siciliano B and Khatib O (eds.) *Handbook of Robotics*, chapter 74. Secaucus, NJ, USA: Springer, pp. 1995–2014. URL https://doi.org/10.1007/978-3-319-32552-1_74. 2nd Edition.

Blocher C, Saveriano M and Lee D (2017) Learning stable dynamical systems using contraction theory. In: *IEEE International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. pp. 124–129. URL https://ieeexplore.ieee.org/document/7992901.

Brehmer J and Cranmer K (2020) Flows for simultaneous manifold learning and density estimation. In: *Neural Information Processing Systems (NeurIPS*. pp. 442–453. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/051928341be67dcba03f0e04104d9047-Paper.pdf.

Bullo F (2024) *Contraction Theory for Dynamical Systems*. 1.2 edition. Kindle Direct Publishing. ISBN 979-8836646806. URL https://fbullo.github.io/ctds.

Chen N, Klushyn A, Paraschos A, Benbouzid D and Van der Smagt P (2018a) Active learning based on data uncertainty and model sensitivity. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. pp. 1547–1554. URL http://doi.org/10.1109/IROS.2018.8593552.

Chen RTQ (2018) torchdiffeq. URL https://github.com/rtqichen/torchdiffeq.

Chen RTQ, Rubanova Y, Bettencourt J and Duvenaud DK (2018b) Neural ordinary differential equations. In: *Neural Information Processing Systems (NeurIPS)*. URL https://proceedings.neurips.cc/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

Dawson C, Gao S and Fan C (2023) Safe control with learned certificates: A survey of neural Lyapunov, barrier, and contraction methods for robotics and control. *IEEE Transactions on Robotics (T-RO)* 39(3): 1749–1767. URL https://doi.org/10.1109/TRO.2022.3232542.

Eklund D and Hauberg S (2019) Expected path length on random manifolds. In: *arXiv preprint*. URL https://arxiv.org/abs/1908.07377.

Falorsi L, de Haan P, Davidson TR and Forré P (2019) Reparameterizing distributions on Lie groups. In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. pp. 3244–3253. URL https://proceedings.mlr.press/v89/falorsi19a.html.

Hauberg S (2019) Only bayes should learn a manifold. URL https://arxiv.org/abs/1806.04994.

Huber L, Billard A and Slotine JJ (2019) Avoidance of convex and concave obstacles with convergence ensured through contraction. *IEEE Robotics and Automation Letters (RA-L)* 4(2): 1462–1469. DOI:http://doi.org/10.1109/LRA.2019.2893676.

Huber L, Slotine JJ and Billard A (2022) Avoiding dense and dynamic obstacles in enclosed spaces: Application to moving in crowds. *IEEE Transactions on Robotics* 38(5): 3113–3132. URL https://ieeexplore.ieee.org/document/9765824.

Hung CM, Zhong S, Goodwin W, Jones OP, Engelcke M, Havoutis I and Posner I (2022) Reaching through latent space: From joint statistics to path planning in manipulation. *IEEE Robotics and Automation Letters (RA-L)* 7(2): 5334–5341. URL http://doi.org/10.1109/LRA.2022.3152697.

Jaffe S, Davydov A, Lapsekili D, Singh A and Bullo F (2024) Learning neural contracting dynamics: Extended linearization and global guarantees. URL https://arxiv.org/abs/2402.08090.

Jouffroy J and I Fossen T (2010) A tutorial on incremental stability analysis using contraction theory. *Modeling, Identification and Control* URL https://doi.org/10.4173/mic.2010.3.2.

Khansari-Zadeh SM and Billard A (2011) Learning stable nonlinear dynamical systems with Gaussian mixture models. *IEEE Transactions on Robotics* 27(5): 943–957. DOI:10.1109/TRO.2011.2159412.

Kingma DP and Ba J (2014) Adam: A method for stochastic optimization. *CoRR* URL https://api.semanticscholar.org/CorpusID:6628106.

Kingma DP and Welling M (2014) Auto-encoding variational Bayes. In: *International Conference on Learning Representations (ICLR)*. URL https://openreview.net/forum?id=33X9fd2-9FyZd.

Koptev M (2023) Implicit distance functions: Learning and applications in robotics DOI:https://doi.org/10.5075/epfl-thesis-10197. URL http://infoscience.epfl.ch/record/305181.

Kozachkov L, Wensing P and Slotine JJ (2023) Generalization as dynamical robustness–the role of riemannian contraction in supervised learning. *Transactions on Machine Learning Research* URL https://openreview.net/forum?id=Sb6p5mcefw.

Lee J (2018) *Introduction to Riemannian Manifolds*. 2 edition. Springer. URL https://sites.math.washington.edu/~lee/Books/RM/.

Lemme A, Meirovitch Y, Khansari-Zadeh M, Flash T, Billard A and Steil JJ (2015) Open-source benchmarking for learned reaching motion generation in robotics. *Paladyn, Journal of Behavioral Robotics* 6(1). URL http://doi.org/10.1515/pjbr-2015-0002.

Lohmiller W and Slotine JJE (1998) On contraction analysis for non-linear systems. *Automatica* 34(6): 683–696. URL https://doi.org/10.1016/S0005-1098(98)00019-3.

Lorraine J and Hossain S (2019) JacNet: Learning functions with structured Jacobians. *INNF Workshop at the International Conference on Machine Learning (ICML)* URL https://invertibleworkshop.github.io/INNF_2019/accepted_papers/pdfs/INNF_2019_paper_10.pdf.

Manchester IR and Slotine JJE (2017) Control contraction metrics: Convex and intrinsic criteria for nonlinear feedback design. *IEEE Transactions on Automatic Control* 62(6): 3046–3053. URL https://doi.org/10.1109/TAC.2017.2668380.

Mandery C, Terlemez O, Do M, Vahrenkamp N and Asfour T (2016) Unifying representations and large-scale whole-body motion databases for studying human motion. *IEEE Transactions on Robotics* 32(4): 796–809. URL https://ieeexplore.ieee.org/document/7506114.

Norcliffe ALI, Bodnar C, Day B, Simidjievski N and Lio P (2021) On second order behaviour in augmented neural ODEs: A short summary. In: *Workshop on the Symbiosis of Deep Learning and Differential Equations at NeurIPS*. URL https://openreview.net/forum?id=XpmaGtI04ki.

Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Kopf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J and Chintala S (2019) Pytorch: An imperative style, high-performance deep learning library. URL https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

Poli M, Massaroli S, Yamashita A, Asama H, Park J and Ermon S (2021) Torchdyn: Implicit models and neural numerical methods in pytorch URL https://github.com/DiffEqML/torchdyn/.

Rana MA, Li A, Fox D, Boots B, Ramos F and Ratliff N (2020a) Euclideanizing flows: Diffeomorphic reduction for learning stable dynamical systems. In: *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, *Proceedings of Machine Learning Research*, volume 120. PMLR, pp. 630–639. URL https://proceedings.mlr.press/v120/rana20a.html.

Rana MA, Li A, Fox D, Boots B, Ramos F and Ratliff N (2020b) Euclideanizing flows: Diffeomorphic reduction for learning stable dynamical systems. In: *Conference on Learning for Dynamics and Control (L4DC)*. pp. 630–639. URL https://proceedings.mlr.press/v120/rana20a.html.

Ravichandar HC and Dani A (2019) Learning position and orientation dynamics from demonstrations via contraction analysis. *Autonomous Robots* 43(4): 897–912. URL https://doi.org/10.1007/s10514-018-9758-x.

Ravichandar HC, Salehi I and Dani A (2017) Learning partially contracting dynamical systems from demonstrations. In: *Conference on Robot Learning (CoRL)*. pp. 369–378. URL https://proceedings.mlr.press/v78/ravichandar17a.html.

Rezazadeh N, Kolarich M, Kia SS and Mehr N (2022) Learning contraction policies from offline data. *IEEE Robotics and Automation Letters* 7(2): 2905–2912. DOI:10.1109/LRA.2022.3145100.

Schaal S, Ijspeert A and Billard A (2003) Computational approaches to motor learning by imitation. *Phil. Trans. R. Soc. Lond. B* 358: 537–547. URL http://doi.org/10.1098/rstb.2002.1258.

Shuster MD (1993) A survey of attitude representation. *Journal of The Astronautical Sciences* 41: 439–517.

Sindhwani V, Tu S and Khansari M (2018) Learning contracting vector fields for stable imitation learning. *arXiv* 1804.04878.

URL https://arxiv.org/abs/1804.04878.

Singh S, Richards SM, Sindhwani V, Slotine JJE and Pavone M (2021) Learning stabilizable nonlinear dynamics with contraction-based regularization. *The International Journal of Robotics Research* 40(10-11): 1123–1150. DOI:10.1177/0278364920949931. URL https://doi.org/10.1177/0278364920949931.

Solà J, Deray J and Atchuthan D (2018) A micro lie theory for state estimation in robotics. *CoRR* abs/1812.01537. URL http://arxiv.org/abs/1812.01537.

Sun D, Jha S and Fan C (2020) Learning certified control using contraction metric. In: *Conference on Robot Learning (CoRL)*. URL http://arxiv.org/abs/2011.12569.

Tabak EG and Turner CV (2013) A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics* 66(2): 145–164.

Tsukamoto H and Chung SJ (2021a) Neural contraction metrics for robust estimation and control: A convex optimization approach. *IEEE Control Systems Letters* 5: 211–216. URL https://doi.org/10.1109/LCSYS.2020.3001646.

Tsukamoto H and Chung SJ (2021b) Robust controller design for stochastic nonlinear systems via convex optimization. *IEEE Transactions on Automatic Control* 66(10): 4731–4746. DOI:10.1109/TAC.2020.3038402.

Tsukamoto H, Chung SJ and Slotine JJE (2021) Contraction theory for nonlinear stability analysis and learning-based control: A tutorial overview. *Annual Reviews in Control* 52: 135–169. URL https://doi.org/10.1016/j.arcontrol.2021.10.001.

Urain J, Ginesi M, Tateo D and Peters J (2020) Imitationflow: Learning deep stable stochastic dynamic systems by normalizing flows. In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 5231–5237. URL http://ras.papercept.net/images/temp/IROS/files/1340.pdf.

Urain J, Tateo D and Peters J (2022) Learning stable vector fields on lie groups. *IEEE Robotics and Automation Letters* 7(4): 12569–12576. DOI:10.1109/LRA.2022.3219019. URL https://ieeexplore.ieee.org/document/9935105.

Xu K, Zhang M, Li J, Du SS, Kawarabayashi KI and Jegelka S (2021) How neural networks extrapolate: From feedforward to graph neural networks. In: *International Conference on Learning Representations (ICLR)*. URL https://openreview.net/forum?id=UH-cmocLJC.

Zhang J, Beik-Mohammadi H and Rozo L (2022) Learning Riemannian stable dynamical systems via diffeomorphisms. In: *Conference on Robot Learning (CoRL)*. URL https://openreview.net/pdf?id=o8dLx8OVcNk.

6 Appendix

6.1 Modulation diagonal matrix D

Based on these requirements, matrix $\boldsymbol{D}$ can be designed according to the following sigmoid function,

$$\Xi(\rho, \nu, \lambda_{init}, \lambda_{end}, k) = \lambda_{init} + \tag{34}$$

$$\frac{\lambda_{end} - \lambda_{init}}{1 + \exp\left(-k\left[\mathcal{S}(\boldsymbol{x}) - \frac{(\rho+\nu)}{2}\right]\right)}, \text{with} \tag{35}$$

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

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. 25, 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 25 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.

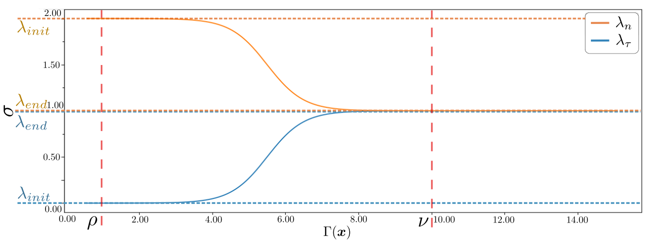


Figure 25. The behavior of the elements of the matrix $\boldsymbol{D}$ in response to distance from obstacle.

6.2 Modulation scaling factor

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, $\mathcal{V}_{\max}$ and $\mathcal{V}_{\min}$, are used to normalize all volumes $\mathcal{V}(\boldsymbol{z})$ as follows,

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

Afterward, α is experimentally chosen to align with the specified distance range defined by ρ and ν in equation 35. This ensures that the scalar field is scaled appropriately, such that:

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