Honda Research Institute EU
Honda Research Institute Europe GmbH
https://www.honda-ri.de/

Learning Deep Dynamical Systems via Stable Neural ODEs

Andreas Sochopoulos, Michael Gienger, Sethu Vijayakumar

2024

Preprint:

This is an accepted article published in IEEE International Conference on Intelligent Robots and Systems (IROS). The final authenticated version is available online at: https://doi.org/10.1109/IROS58592.2024.10801826
Copyright 2024 IEEE

Learning Deep Dynamical Systems using Stable Neural ODEs

Andreas Sochopoulos[1,2] Michael Gienger[2] and Sethu Vijayakumar[1,3]

***Abstract*— Learning complex trajectories from demonstrations in robotic tasks has been effectively addressed through the utilization of Dynamical Systems (DS). State-of-the-art DS learning methods ensure stability of the generated trajectories; however, they have three shortcomings: a) the DS is assumed to have a single attractor, which limits the diversity of tasks it can achieve, b) state derivative information is assumed to be available in the learning process and c) the state of the DS is assumed to be measurable at inference time. We propose a class of provably stable latent DS with possibly multiple attractors, that inherit the training methods of Neural Ordinary Differential Equations, thus, dropping the dependency on state derivative information. A diffeomorphic mapping for the output and a loss that captures time-invariant trajectory similarity are proposed. We validate the efficacy of our approach through experiments conducted on a public dataset of handwritten shapes and within a simulated object manipulation task.**

I. Introduction

Learning from Demonstrations (LfD) is a methodology that allows robots to learn complex behaviours by closely observing and emulating human or expert demonstrations [1], that would otherwise be too cumbersome to compute or hand code. By leveraging this method, robots can bridge the gap between human expertise and machine capabilities, facilitating rapid skill acquisition and adaptation. Replicating human motions can also greatly benefit the collaboration capabilities of the robot in Human Robot Interaction (HRI) tasks and act in more predictable ways for humans. Moreover, the flexibility of the framework, allows the robot to learn how humans move in order to use it as a predictive model rather than replicate their movement. Creating surrogate models for humans has shown good potential in HRI [2] and exoskeleton assistance adaptation tasks [3].

Human motions often exhibit complex patterns, smoothness, and stability. However, encoding these constraints on learnt models is non-trivial. Dynamical Systems (DS) provide a versatile framework for encoding such trajectories with intrinsic structures that, if properly constructed, ensure stability, smoothness, and the ability to capture intricate spatio-temporal dependencies.

Stability in learning methods is typically addressed in terms of globally stable attractors, limiting the system to single-goal-driven motions. Although most tasks can be split up in multiple goal-driven motions [4], identifying the hierarchy of subtasks and their hyperparameters is non-trivial. DS with multiple attractors [5] and DS whose trajectories are constrained to move in a subset of the phase space can represent more complex behaviours and increase the variety of tasks a DS can achieve. Moreover, many existing approaches in LfD assume that the state whose dynamics are being captured is readily available for measurement. However, in many robotic tasks, this assumption does not hold true due to either a lack of sensing capabilities or a mismatch between sensing and state modalities. For example, in tasks where only vision is available as a sensing modality, extracting the end effector pose from images adds complexity and can compromise accuracy. Additionally, most methods rely on state derivative information [6]–[8], such as position, velocity, and acceleration, treating the learning process as a regression problem. This requirement adds complexity to the demonstration acquisition process, necessitating additional hardware and data processing. The quality of the data is crucial as well, as the system learns to reproduce demonstrated trajectories with their timing intact. Speeding up demonstrations can lead to a completely different learned model, making reproducibility challenging and complicating the acquisition process further.

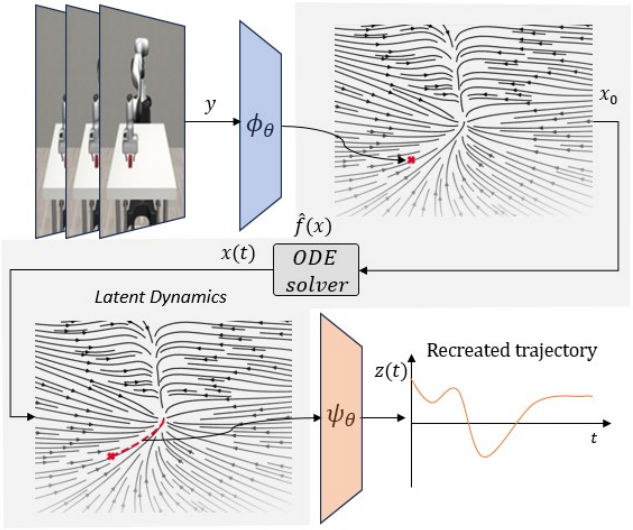


Fig. 1: End-to-End learning framework for Dynamical Systems using stable Neural ODEs

A. Related Works

Dynamic Movement Primitives (DMPs) [9] were among the first approaches to utilize DS for learning robot motions. The DS learned through DMPs typically consist of a stable linear component and a nonlinear forcing term composed of Radial Basis Functions (RBFs) with weights adapted to closely match demonstrated trajectories. Subsequently, the

[1] School of Informatics, The University of Edinburgh, Edinburgh, UK
[2] Honda Research Institute Europe GmbH, Offenbach am Main, Germany
[3] The Alan Turing Institute, London, U.K.

first approach to approximate the entire DS with stability guarantees was SEDS [6], which employed Gaussian Mixture Models (GMMs) as function approximators for the dynamics. Lyapunov stability is ensured for a single goal state by imposing constraints on the weights of the GMM. However, the linear nature of DMPs and the constraints of SEDS on the weights of the GMM limit their representational capabilities, allowing only for the learning of simple motions evolving in low-dimensional phase spaces.

The trade-off between accuracy and stability has been addressed in later methods [7], [10], which assume an unconstrained and potentially unstable part of the dynamics alongside a corrective term that ensures global stability, in a Lyapunov sense, of a designated attractor. Additionally, other methods have achieved increased accuracy by employing diffeomorphisms [8], [11], [12]. Specifically, these methods construct a simple and provably stable DS in a latent space, which is then mapped through parameterized diffeomorphisms in the output space. This learning paradigm updates only the weights of the parameterized diffeomorphism rather than the state mapping of the DS. The stability of the output space system is guaranteed since it is diffeomorphic to the latent space system which is stable by design. Lastly, some approaches have capitalized on the non-Euclidean nature of the data encountered in robot applications and have introduced DS that evolve in manifolds [8], [12].

DS offer spatiotemporal representations that have been utilized in machine learning as well. Neural ODEs [13] have emerged as a method to encode the hidden layers of Residual Networks as a continuous time ODE. They are a class of architectures that combine neural networks with dynamical systems and thus naturally fit into the problem of learning DS for robot and control applications [14]–[16].

B. Contributions

In this paper, we use Neural ODEs to represent and train DS with deep state mappings. Since they lack intrinsic mechanisms to ensure the stability of the latent dynamics, we propose a corrective signal that builds upon [10] and [7].

The contributions of this paper are: a) a Neural ODE framework for learning DS with *multiple attractors* and proven stability guarantees, b) a novel loss that can capture trajectory similarity in the phase space instead of the time domain, c) formulation of the necessary components to correctly set the attractors in the latent space through diffeomorphic output mappings and d) enabling LfD with *multiple feedback modalities* and without the need for *state derivatives*. Overall, the aim of this paper is to introduce a flexible DS learning paradigm that requires minimal overhead, is versatile and can capture intricate motions. These contributions are made clear in Table I where we also compare them with the characteristics of other similar methods.

II. Backgournd

A. Dynamical Systems

The task that the majority of DS learning methods consider is to learn a continuous time autonomous dynamical system given by the equation:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}(t)),\ t \in T \tag{1}$$

where t is time, T a time interval, $\boldsymbol{x}(t) \in \boldsymbol{X} \subset \boldsymbol{R}^{n_x}$ is the state of the DS and $\boldsymbol{f} : \boldsymbol{X} \longrightarrow \boldsymbol{X}$ a potentially nonlinear vector field to be learnt. In the particular case where this DS acts as a trajectory generator for a robotic system, $\boldsymbol{x}(t)$ represents the desired configuration of the robot in addition to the state of any other agent or object that interacts with it.

This formulation assumes that the state $\boldsymbol{x}$ of the DS is available for measurement. Furthermore, conventional DS learning approaches [6]–[8] necessitate access to state derivative information for the supervised learning of the underlying function $\boldsymbol{f}$. This prerequisite presents significant challenges when endeavoring to learn dynamics within latent spaces or when derivative information is not readily accessible. To illustrate, consider a scenario where human joint trajectories, obtained from vision-based techniques [17], serve as the training data. In such cases, obtaining derivative information typically entails numerical differentiation, a process prone to yielding sub-optimal outcomes and contingent upon factors such as sampling frequency and temporal regularity.

B. Neural ODEs

In light of these considerations, we opt for Neural ODEs [13] to model DS. This choice is motivated by their capability to address the aforementioned challenges with minimal adjustments.

Definition 1 (Neural ODE): Assume an observation $\boldsymbol{y} \in \boldsymbol{Y}$ that could potentially be any sensing modality. A Neural ODE is defined as the following ODE system:

$$\begin{cases} \boldsymbol{x}(t_0) = \boldsymbol{\phi_\theta}(\boldsymbol{y}) & (input\ mapping) \\ \dot{\boldsymbol{x}} = \boldsymbol{f_\theta}(\boldsymbol{x}) & (latent\ state\ dynamics) \\ \boldsymbol{z}(t) = \boldsymbol{\psi_\theta}(\boldsymbol{x}(t)) & (output\ mapping) \end{cases} \tag{2}$$

where $\boldsymbol{x} \in \boldsymbol{X} \subset \boldsymbol{R}^{n_x}$ is the state variable of the ODE, $\boldsymbol{z}(t) \in \boldsymbol{Z} \subset \boldsymbol{R}^{n_z}$ is the output variable $\boldsymbol{\phi_\theta}(\cdot) : \boldsymbol{R}^{n_y} \longrightarrow \boldsymbol{R}^{n_x}$ and $\boldsymbol{\psi_\theta}(\cdot) : \boldsymbol{R}^{n_x} \longrightarrow \boldsymbol{R}^{n_z}$ are the input and output functions. The vector field $\boldsymbol{f_\theta} : \boldsymbol{R}^{n_x} \longrightarrow \boldsymbol{R}^{n_x}$ determines the evolution of the dynamics and together with the input and output functions, it is a network with learnable parameters $\boldsymbol{\theta}$. The ODE is solved along an arbitrary time interval $T = [t_0,\ t_f]$ where t_0 and t_f are the initial and final time respectively.

Neural ODEs present a framework conducive to the acquisition of deep DS operating within a latent variable $\boldsymbol{x}$. This inherent characteristic renders them particularly advantageous in applications within robotics and control domains where DS play a pivotal role. Despite their ability in capturing intricate behaviors and processes, Neural ODEs lack an intrinsic mechanism to ensure that the solution of the learned ODE remains bounded, thus mitigating the risk of divergence for specific initial values within the phase space $\boldsymbol{X}$. In the realm of robotics, assuring stability is imperative as it correlates closely with safety and predictability.

TABLE I: Comparison of DS Learning Methods based on their capacity to learn different types of DS. The first column shows the dependence of the learning methods on state derivative information. The second and third column indicate the ability of the model to learn DS with multiple attractors and DS that live in latent spaces respectively. Lastly stability is referred to the Lyapunov stability of the learnt DS. *: Although multiple attractors can be potentially handled by [8], no method to achieve this is given.

	State Derivative Free	Multiple attractors	Multiple Input Modalities	Stable
CLFDM [7]	–	–	–	✓
Deep Dynamics [10]	✓	–	✓	✓
Euclideanizing Flows [8]	–	✓*	–	✓
StableNODE (Ours)	✓	✓	✓	✓

There have been various attempts [18], [19] to address this concern; however, the majority fall short in providing stability guarantees spanning the entirety of the phase space. Typically, assurances are confined to regions encountered during training, thereby offering weaker guarantees than established DS learning methods [6]–[8].

III. Stable Neural ODEs for DS learning

In this section we present StableNODEs, which build upon techniques that ensure stability in DS.

Definition 2 (StableNODE): Given an observation $\boldsymbol{y}$, a StableNODE is defined by:

$$\begin{cases} \boldsymbol{x}(t_0) = \boldsymbol{\phi_\theta}(\boldsymbol{y}) \\ \dot{\boldsymbol{x}} = \hat{\boldsymbol{f}}(\boldsymbol{x}) \\ \boldsymbol{z}(t) = \boldsymbol{\psi_\theta}(\boldsymbol{x}(t)) \end{cases} , t \in T \tag{3}$$

where $\hat{\boldsymbol{f}}(\boldsymbol{x}) = \boldsymbol{f_\theta}(\boldsymbol{x}) + \boldsymbol{u}(\boldsymbol{x})$, with:

$$\boldsymbol{u}(\boldsymbol{x}) = \begin{cases} 0 & , L(\boldsymbol{x}) \leq 0 \\ -\frac{\nabla V_{\boldsymbol{\theta}}(\boldsymbol{x}) L(\boldsymbol{x}) + \epsilon s L(\boldsymbol{x}) \boldsymbol{f_\theta}(\boldsymbol{x})}{|\nabla V_{\boldsymbol{\theta}}(\boldsymbol{x})|^2 + \epsilon} & , 0 < L(\boldsymbol{x}) < \frac{1}{s} \\ -\frac{\nabla V_{\boldsymbol{\theta}}(\boldsymbol{x}) L(\boldsymbol{x}) + \epsilon \boldsymbol{f_\theta}(\boldsymbol{x})}{|\nabla V_{\boldsymbol{\theta}}(\boldsymbol{x})|^2 + \epsilon} & , L(\boldsymbol{x}) \geq \frac{1}{s} \end{cases} \tag{4}$$

where $L(\boldsymbol{x}) = \nabla V_{\boldsymbol{\theta}}(\boldsymbol{x}) \boldsymbol{f_\theta} + \alpha V_{\boldsymbol{\theta}}(\boldsymbol{x})$, ϵ, α and s are positive constants and $V_{\boldsymbol{\theta}}(\cdot)$ is a learnable *energy function*. The rest of the notation is consistent with Definition 1. Note that although the energy, input, output and state mappings are different networks, they are all trained jointly according to the gradients of the Neural ODE loss.

The rationale behind function $\hat{\boldsymbol{f}}$ is that it mirrors the nominal function $\boldsymbol{f_\theta}$ when $L(\boldsymbol{x}) \leq 0$. Here, the condition $L(\boldsymbol{x}) \leq 0$ signifies that the nominal dynamics have solutions with exponentially decreasing energy according to $V_{\boldsymbol{\theta}}$, thus requiring no alteration. However, when this condition is not met, the nominal dynamics are adjusted in a manner that ensures solutions with decreasing energy.

The formulation of function $\hat{\boldsymbol{f}}$ expands upon the ideas presented in [7], [10], where energy functions take the form of Lyapunov functions. Notably, function $\hat{\boldsymbol{f}}$ and the functions proposed in [10] coincide when $\epsilon = 0$. In [7], the Lyapunov function is represented as a sum of asymmetric quadratic terms, while in [10], it takes the form of an Input Convex Neural Network (ICNN), resulting in convex Lyapunov functions characterized by a single global minimum. By employing function (4), we broaden the scope to include energy functions that need not necessarily be convex. While still required to serve as candidate Lyapunov functions, they are permitted to exhibit critical points other than the attractor. This is a key characteristic that lets us define energy functions with multiple global minima (or attractors) and incorporate repulsive energy fields for specific regions of the phase space as seen in Fig. 3. The easiest way to construct such energy functions is by combining many attractive and repulsive energy functions [20], with some or all of them being learnable functions. However, it is impossible to combine energy functions to a function that exhibits no spurious critical points [20], which can render the stabilizing signal used in [10] infinite. Although StableNODEs don't guarantee the avoidance of spurious saddle points and local minima, the issue of undefined dynamics in their vicinity is resolved, ensuring computational stability.

The stability guarantees of StableNODEs are presented in the following theorem:

Theorem 1: *Assume the dynamics:*

$$\dot{\boldsymbol{x}} = \hat{\boldsymbol{f}}(\boldsymbol{x}) \tag{5}$$

where $\hat{\boldsymbol{f}}(\boldsymbol{x})$ is given by the sum of a nominal model and the corrective signal (4). Assume a finite set of desired attractors $\boldsymbol{X}_e$ and a candidate energy function that satisfies $V_{\boldsymbol{\theta}}(\boldsymbol{x}) > 0,\ \forall \boldsymbol{x} \in \boldsymbol{X} \setminus \{\boldsymbol{X}_e\}$ and $V_{\boldsymbol{\theta}}(\boldsymbol{x}) = 0,\ \forall \boldsymbol{x} \in \boldsymbol{X}_e$. Assume $\boldsymbol{X}_c$ is a set of any critical points of $V_{\boldsymbol{\theta}}$ that do not belong in $\boldsymbol{X}_e$. Then the solutions of (5) converge asymptotically to the set $\boldsymbol{E} = \{\boldsymbol{x} \in \boldsymbol{X} | V_{\boldsymbol{\theta}}(\boldsymbol{x}) \leq \frac{1}{s\alpha}\} \cup \boldsymbol{X}_c$.

Proof: For the proof see the Appendix

A. Attractors in the output space

The latent space of Neural ODEs often lacks a direct physical interpretation within the context of the problem they are applied to. The most significant variables, in many cases, are the output variables, as they are the ones that can be interpreted by the user. For instance, a StableNODE could be utilized to learn a DS capable of replicating the trajectory of an end effector of a manipulator. In this scenario, the latent space of the StableNODE holds no inherent physical meaning to the problem, as both functions $\boldsymbol{\phi_\theta}(\cdot)$ and $\boldsymbol{\psi_\theta}(\cdot)$ are learned in an end-to-end fashion. However, the output corresponds to the positions of the robot's end effector. Consequently, a commonly acceptable choice for the attractor of the learned DS would be the final point in the demonstrated trajectory. Nevertheless, as the attractor for the StableNODE exists in the latent space, there is no direct way to set it to a desired value, and ensuring that $\boldsymbol{\psi_\theta}(\boldsymbol{0})$ coincides with that point cannot be guaranteed. Therefore, it is imperative to devise a mechanism that maps an attractor set $\boldsymbol{Z}_e$ in the

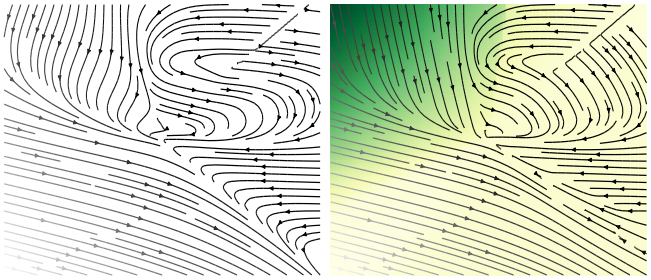


Fig. 2: (a) The vector field of the nominal function $\boldsymbol{f_\theta}$ for a 2 dimensional latent space and (b) The vector field of the sum of $\boldsymbol{f_\theta}$ and the corrective term $\boldsymbol{u}$. Dark green regions denote the areas of the phase space where $L(\boldsymbol{x}) > 0$.

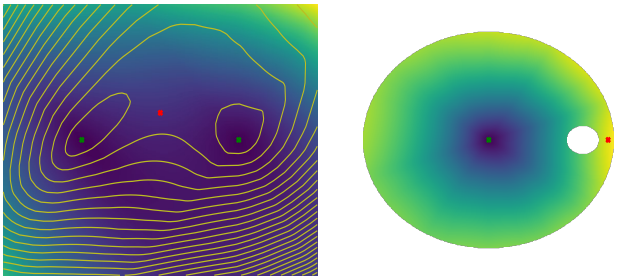


Fig. 3: (a) A 2-dimensional energy function with two target points shown in green. The red point is a critical point of $V_{\boldsymbol{\theta}}$ and in this case is either a local maximum or a saddle point. This energy function can be useful if the DS has to learn reaching or placing motions with 2 goal positions, (b) A single-target energy function. The energy function is build to be maximal on the boundaries of an outer circle and of an inner circle restricting thus the resulting trajectories. It is impossible to create such functions that do not exhibit other critical points than the target (target shown in green, critical point in red).

output space uniquely to a set $\boldsymbol{X}_e$ in the latent space and transfers the stability guarantees from the latent to the output space.

In order to specify $\boldsymbol{X}_e$, a continuous mapping that can map $\boldsymbol{Z}_e$ to $\boldsymbol{X}_e$ and is invertible, such that $\boldsymbol{X}_e$ can be mapped directly to $\boldsymbol{Z}_e$, is needed. Mappings $\boldsymbol{\psi} : \boldsymbol{R}^{n_x} \longrightarrow \boldsymbol{R}^{n_x}$ that satisfy these conditions are called bijective and if used as output mappings, they allow determining $\boldsymbol{X}_e$ as $\boldsymbol{X}_e = \{\boldsymbol{\psi}^{-1}(\boldsymbol{z}),\ \forall \boldsymbol{z} \in \boldsymbol{Z}_e\}$.

This raises the question of how to construct learnable bijective mappings that are expressive enough to capture complex behaviors. To address this challenge, we construct our output mapping as a synthesis of n simpler bijective layers, denoted as $\boldsymbol{\psi_\theta} = \boldsymbol{\psi}_1 \circ \boldsymbol{\psi}_2 \circ \ldots \circ \boldsymbol{\psi}_n$, where each bijective layer $\boldsymbol{\psi}_k : \mathbb{R}^{n_x} \longrightarrow \mathbb{R}^{n_x}$ is implemented as a coupling layer [21]. Let $\boldsymbol{x} \in \mathbb{R}^{n_x}$ be the input to a coupling layer $\boldsymbol{\psi}_k$. The input vector is first divided into two vectors $\boldsymbol{x}^\alpha \in \mathbb{R}^{\lfloor \frac{n_x}{2} \rfloor}$ and $\boldsymbol{x}^\beta \in \mathbb{R}^{\lceil \frac{n_x}{2} \rceil}$. Subsequently, one of the two parts, with the order reversed in the subsequent layer, undergoes a scaling and translation operation. The output of layer k is given by:

$$\boldsymbol{\psi}_k(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{x}^\alpha \\ \boldsymbol{x}^\beta \odot exp(\boldsymbol{s}_k(\boldsymbol{x}^\alpha)) + \boldsymbol{t}_k(\boldsymbol{x}^\alpha) \end{bmatrix} \tag{6}$$

where $\odot$ denotes the Hadamard product $\boldsymbol{s}_k : R^{\lfloor \frac{n_x}{2} \rfloor} \longrightarrow R^{\lfloor \frac{n_x}{2} \rfloor}$ and $\boldsymbol{t}_k : R^{\lfloor \frac{n_x}{2} \rfloor} \longrightarrow R^{\lfloor \frac{n_x}{2} \rfloor}$ are learnable scaling and translation functions respectively, implemented as feed-forward neural networks. This mapping and any composition of finitely many such mappings is guaranteed to be bijective. Furthermore, the inverse is straightforward and cheap to compute. A major limitation, however, of having a diffeomorphic mapping as an output layer is that the dimensionality of the latent and the output space needs to be the same. The readers are referred to [21] for more details of this mapping and to [8] for the stability properties of the output space dynamical system.

B. Average Hausdorff distance as a loss function

Neural ODEs have predominantly been employed in classification problems [13], [18], where the ODE is solved forward in time, yielding the final time value as the model's prediction. Consequently, the loss function becomes a function of $\boldsymbol{x}(t_f)$, independent of intermediate trajectory values. However, when Neural ODEs are utilized to model DS, such as in robot planning and control or human motion prediction, the loss function typically encompasses the entire trajectory $\boldsymbol{x}_{t_0:t_f}$. This distinction impacts the selection of the loss function and the approach to computing its gradients with respect to the trainable parameters. The former will be addressed in this subsection, while the latter will be discussed in the subsequent section.

Demonstrations used to train robots in acquiring skills are typically untimed, and tasks are often non-time-critical. The primary objective is for the robot to reproduce the demonstrated trajectory, regardless of whether it executes it slightly faster or slower. However, a Mean Squared Error (MSE) loss, comparing the simulated and actual trajectories, not only measures trajectory similarity in phase space but also captures timing discrepancies. Consequently, even if a trajectory is identical to the demonstrated one in phase space, it could incur a high MSE loss due to timing differences. This phenomenon is visible in Fig. 4. Moreover, besides deviating from our objective, an MSE loss imposes strict training criteria, necessitating more training iterations and increased network parameters.

Furthermore, considering $\hat{\boldsymbol{f}}$, there's a significant hyper-parameter to be determined, with substantial implications for the learning process and its outcome. The parameter α dictates the rate of Lyapunov function decay and is selected a priori. However, since the Lyapunov function is unknown beforehand, choosing α is not straightforward. Large values can render the optimization problem challenging to solve, as it may be improbable to find a Lyapunov function that decreases exponentially with rate α for a $\hat{\boldsymbol{f}}$ perfectly tracking desired trajectories. Nevertheless, by minimizing a trajectory similarity loss in the phase space, we afford $\hat{\boldsymbol{f}}$ and $V_{\boldsymbol{\theta}}$ more flexibility to be learned while adhering to the imposed constraints. For these reasons, we opt to employ the Average Hausdorff Distance (AHD) [22] as the loss function between predicted and actual trajectories.

Consider two unordered non-empty sets $\hat{\boldsymbol{X}}$ and $\boldsymbol{X}_d$ and

a distance metric $d(\boldsymbol{x}, \boldsymbol{x}_d)$ between two points $\boldsymbol{x} \in \hat{\boldsymbol{X}}$ and $\boldsymbol{x}_d \in \boldsymbol{X}_d$. The distance metric could be any metric, however in our case the Euclidean distance in the phase space is used. Note that the two sets do not have to be of the same size. For problems addressed in this paper, $\hat{\boldsymbol{X}}$ is a trajectory generated by solving the StableNODE forward in time, while $\boldsymbol{X}_d$ is a demonstrated trajectory. Then the AHD is defined as:

$$L_H = \frac{1}{|\hat{\boldsymbol{X}}|} \sum_{\boldsymbol{x} \in \hat{\boldsymbol{X}}} \min_{\boldsymbol{x}_d \in \boldsymbol{X}_d} d(\boldsymbol{x}, \boldsymbol{x}_d) + \frac{1}{|\boldsymbol{X}_d|} \sum_{\boldsymbol{x}_d \in \boldsymbol{X}_d} \min_{\boldsymbol{x} \in \hat{\boldsymbol{X}}} d(\boldsymbol{x}, \boldsymbol{x}_d) \tag{7}$$

Loss (7), if implemented within an autograd framework, like PyTorch, is differentiable for all $(\boldsymbol{x}, \boldsymbol{x}_d)$ although it is not strictly continuous. Notice how the definition does not require the two sets $\hat{\boldsymbol{X}}$ and $\boldsymbol{X}_d$ to be ordered. Trajectories can be considered as sets ordered by time, however as mentioned in the previous paragraphs, the trace of the trajectories on the phase plane is of importance. Moreover, the two sets don't have to be of the same size in order to calculate the loss, which makes it easier to calculate using the solution of an adaptive solver and irregularly sampled demonstrations.

C. Computing loss gradients

Training a Neural ODE is equivalent to solving an optimization problem of the form:

$$\begin{aligned} & \min_{\theta} \; L[\boldsymbol{x}(t, \boldsymbol{\theta})] \\ s.t. \quad & \dot{\boldsymbol{x}} = \hat{\boldsymbol{f}}(\boldsymbol{x}) \\ & \boldsymbol{x}(t_0) = \boldsymbol{\phi}_{\boldsymbol{\theta}}(\boldsymbol{y}) \end{aligned} \tag{8}$$

with respect to $\boldsymbol{\theta}$, where $L[\boldsymbol{x}(t, \boldsymbol{\theta})] = \int_{t_0}^{t_f} l(\boldsymbol{x}(t, \boldsymbol{\theta}))\, dt + l_f(\boldsymbol{x}(t_f, \boldsymbol{\theta}))$ represents a loss functional. Here, l and l_f denote arbitrary instantaneous and final costs, respectively. These losses can take the form of similarity metrics, such as (7), comparing the Neural ODE-generated trajectory with demonstrated trajectories, or encode any other desirable properties. If the demonstrations include state derivatives, they could be utilized as labels for training $\hat{\boldsymbol{f}}$ in a regression fashion. However, the optimization problem (8) does not assume availability of state derivatives and allows for more intricate costs to be minimized through the functions l and l_f. Subsequently, solving (7) is challenging, as it requires computing gradients throughout the solution of the latent ODE.

The derivatives of the functional (8) can be computed using the *state adjoint method* [23], which involves solving the latent ODE forward in time and the adjoint ODE backward in time [13]. This approach offers significant benefits in terms of memory cost for calculating derivatives compared to using ODE solvers implemented within an auto-differentiation framework. However, solving two ODEs, as required by the *state adjoint method*, can be considerably slower when evaluating the functional (8) over a large time period.

For computational efficiency reasons, auto-differentiation through the ODE solver is used as the default gradient calculation method for (8) throughout this paper, unless otherwise mentioned.

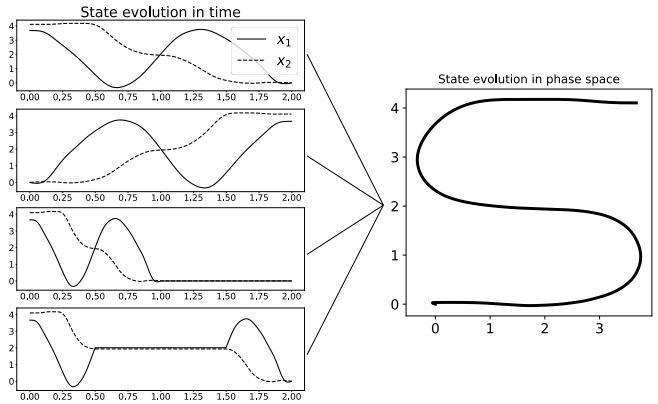


Fig. 4: Four trajectories with time misalignments that correspond to the same phase space trajectory.

IV. Experiments

A. Results on the LASA dataset

The effectiveness of StableNODEs in learning demonstrated trajectories is evaluated using the benchmark LASA dataset [6]. This open-source dataset, comprises 30 handwritten shapes, with seven different demonstrations provided for each shape, resulting in a total of 210 trajectories. The performance of Euclideanizing Flows (EF) [8] and Deep Stable Dynamics (DSD) [10] is also assessed and compared against StableNODEs.

To capture the expressiveness of the latent space formulation utilized in StableNODEs, we adopt the identity maps for both input and output mappings. The nominal function $\boldsymbol{f}_{\boldsymbol{\theta}}$ is modeled using a feedforward network, while the Lyapunov function is represented by an ICNN. We employ the training loss (7) and obtain solutions of the ODEs using the *Dopri5* solver implemented in PyTorch with the package *torchdiffeq* [13]. The parameters for the corrective term are set as follows: $\epsilon = 1 \times 10^{-5}$, $\alpha = 1 \times 10^{-3}$, and $s = 20$. For DSD, we utilize the same nominal model as for StableNODE, set ϵ to 0, use the Mean Squared Error (MSE) loss, and obtain solutions using the *Euler* solver. EFs are implemented following the method described in [8].

All three methods are evaluated using the Dynamic Time Warping Distance (DTWD) and the Discrete Frechet Distance (FD), both of which are time-invariant measures of time-series similarity. These metrics effectively capture the similarity between the demonstrated and recreated trajectories in the phase space and are commonly employed to evaluate DS learning methods [8], [12].

As depicted in Fig. 5, StableNODE consistently outperforms the other two DS learning methods across multiple shapes. StableNODE demonstrates a lower median and standard deviation of DTWD scores throughout the dataset, while DSD exhibits a lower median Frechet score. Notably, both StableNODE and DSD outperform EF, which could potentially be attributed to the fact that velocity information

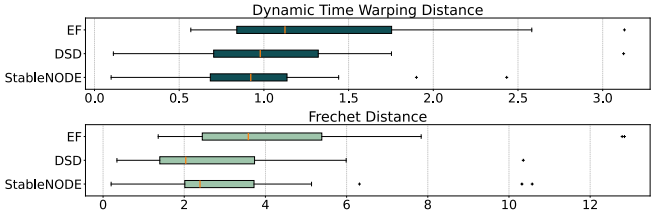


Fig. 5: Dynamic Time Warping Distance and Frechet Distance of Euclideanizing Flows (EF), Deep Stable Dynamics (DSD) and StableNODEs.

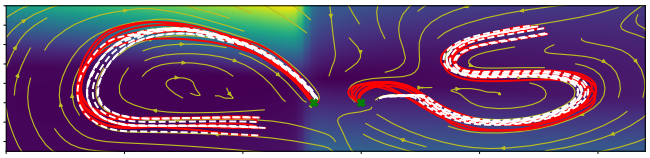


Fig. 6: Demonstrated trajectories (solid red line) and DS generated trajectories (white dashed line) of a DS with two target points.

required for EF's training was obtained through finite differentiation. It is also worth mentioning that data resampling and smoothing operations were necessary to achieve the performance with EF, whereas raw trajectories were used for learning with DSD and StableNODE.

Although DSD and StableNODE demonstrate similar performance, it is conceivable that StableNODEs would outperform DSD in scenarios with smaller sampling frequencies, irregularly spaced data, or a mixture of fast and slow modes in the trajectories, owing to the utilization of adaptive solvers.

The capability of StableNODE to model DS with multiple attractors is also assessed by combining the translated demonstrations from two different shapes, as illustrated in Fig. 6. To capture this behavior, an energy function of the form $V_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma_\gamma(\boldsymbol{x})V_1(\boldsymbol{x}-\boldsymbol{x}_1^*) + (1-\sigma_\gamma(\boldsymbol{x}))V_2(\boldsymbol{x}-\boldsymbol{x}_2^*)$ is employed, where V_1 and V_2 are represented by ICNNs, $\boldsymbol{x}_1^* = [0,\ 0]^T$ and $\boldsymbol{x}_2^* = [0,\ -0.2]^T$ denote the two attractors, and $\sigma_\gamma(\boldsymbol{x})$ represents the sigmoid function with a positive scale factor $\gamma = 70$.

From Fig. 6, it is evident that StableNODE can generate trajectories from either shape that converge to the correct attractor. However, performance is slightly compromised due to the overlapping of the two Lyapunov functions; this could potentially be minimized by placing the attractors further apart.

B. Image-to-trajectory generation for a simulated picking task

The efficacy of StableNODEs as generative models for image-to-trajectory generation is evaluated in a simulated picking task, involving a Franka Emika robot and a red cube placed on a table. This simulation environment is part of Robosuite [24] and has been used in other works as a benchmark for evaluating vision-based policy learning techniques [25]. We tackle a similar problem, that of generating a trajectory for the end-effector of the robot using a single RGB image of the initial state of the workspace, as can be seen in Fig. 8. The position of the cube is considered unknown both during training and inference and it is sampled uniformly within a 0.03 by 0.02 rectangle at the beginning of each trial.

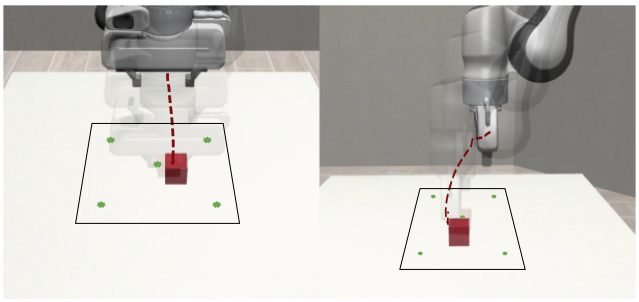


Fig. 7: Generated end-effector trajectory using a RGB image. The green stars represent the 5 attractors of the DS and the black box denotes the surface within which the cube can be spawned.

Demonstrations of successful executions of the task from proficient humans were used for training a StableNODE. 120 successful and balanced, with respect to the position of the cube, demonstrations were used for the training of the StableNODE.

The StableNODE used consists of a 4 layer convolutional input network and an output mapping of three stacked coupling layers given by (6). As depicted in Fig. 8, during a training a step, a batch of N_B demonstrations of length N_S is randomly sampled. The RGB images $\boldsymbol{y} \in \boldsymbol{R}^{84\times84\times3}$ collected at the initial time sample of the batched trajectories, are used as inputs to the input network $\boldsymbol{\phi_\theta}(\cdot)$, which in turn produces a batch of points $\boldsymbol{x}_0$ in the latent space of the StableNODE. These points are then used as initial values to an ODE solver according to eq. (2). The solution of the ODE and $\boldsymbol{x}_0$ concurrently pass through the output mapping to produce a generated trajectory $\boldsymbol{z}_{t_0:N_S}$ and an image-to-robot position prediction $\boldsymbol{z}_{t_0}$. The training loss is then calculated as:

$$L = L_H\left[\boldsymbol{z}_{t_0:t_{N_S}}, \boldsymbol{z}^d_{t_0:t_{N_S}}\right] + k\cdot||\boldsymbol{z}_{t_0} - \boldsymbol{z}^d_{t_0}||^2 \qquad (9)$$

We found that the second term of eq. (9) helps align the initial position of the generated trajectory with that of the end effector, since end effector position information is not assumed to be available. The latent dynamics $\hat{\boldsymbol{f}}$, together with an energy function $V_{\boldsymbol{\theta}}$, implemented as a mixture of ICNNs, are learnt by minimizing L_H. The hyperparameters used in our experiments are $k = 15,\ N_B = 120,\ N_S = 25,\ \epsilon = 10^{-4},\ s = 10$ and $\alpha = 1$. The output $z \in \boldsymbol{R}^6$ of the StableNODE is the end-effector position and velocity.

Since the cube position is unknown, it cannot be used as an attractor, which means that grasping the cube is part of the transient behavior of the DS. We hypothesize that introducing attractors located near the edges and center of the rectangular area the cube lies in, the DS can be steered to any location the cube is in. In order to achieve this, we introduce 5 attractors: $\boldsymbol{z}_0^* = [0, 0, 0.82, 0, 0, 0]^T$ and $\boldsymbol{z}_i^* = [\pm0.025, \pm0.015, 0.82, 0, 0, 0]^T, i = 1, 2, 3, 4$, as seen in Fig. 7, and 5 ICNNs $V_i,\ i = 0, 1, ..., 5$ associated with each attractor. An energy function is constructed as

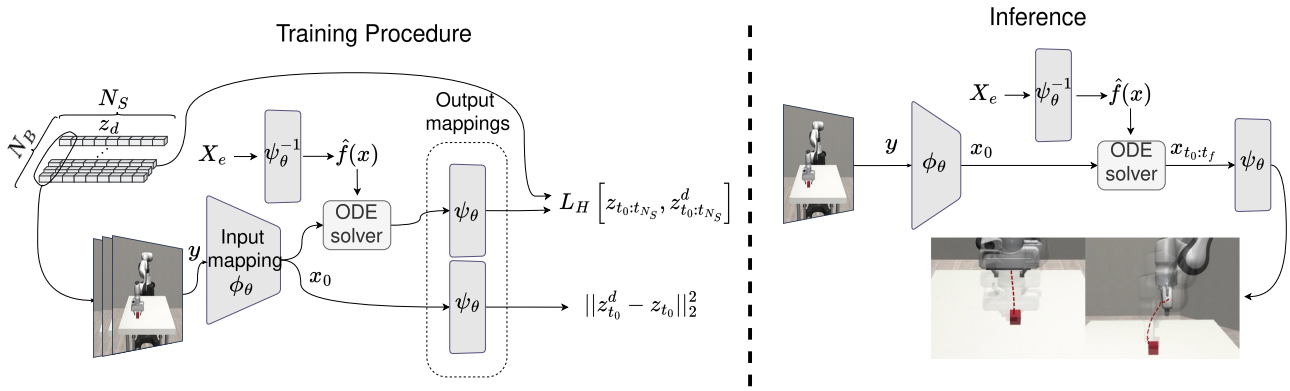


Fig. 8: The end-to-end training process for a StableNODE to generate trajectories from a single RGB image (left) and the trajectory generation process (right). N_B denotes the number of training batches and N_S the size of the training trajectories.

$$V = \prod_{i=0}^{4} V_i(\boldsymbol{x} - \boldsymbol{\psi}_{\boldsymbol{\theta}}^{-1}(\boldsymbol{z}_i^*)) \ .$$

During inference an image is fed to the trained StableNODE, which in turn produces a trajectory for the end-effector. We evaluated the ability of the generated trajectories to effectively solve the task of picking the cube over 1000 trials, with the position of the cube sampled uniformly from a 0.03 by 0.02 rectangle at every trial. The task was considered complete once the cube was lifted from the table. Furthermore, the control of the end-effector is considered to be out of scope for this problem and is thus implemented using the robot's and the cube's position.

Our method achieved a success percentage of 91.27%, despite the tracking errors introduced by the robot controller. In the failed attempts, although the robot failed to grasp the cube, its behaviour was stable and it always approached one of the attractors. Furthermore, a StableNODE trained on only 6 demonstrations was evaluated in order to assess the generalization capabilities of the method in the absence of data for a large part of the phase space. The method was successful 73.53% of the trials, exhibiting good behaviour for cube locations close to the ones seen in training. Overall, although this method could never compete with state-of-the-art visuomotor policies, the results on this tasks show that StableNODEs can encode more than one mode of behaviour, as single attractor methods would.

V. Conclusions

In this study, we introduce a framework utilizing NeuralODEs for the learning of stable DS. Our approach augments Neural ODEs by incorporating a corrective function, ensuring provable stability across the entire phase space, even in the presence of multiple attractors. Additionally, we propose an output mapping mechanism comprising learnable diffeomorphisms, facilitating the definition and transfer of attractors between the output and latent spaces. We employ the Average Hausdorff Distance (AHD) as the training loss, as it offers a time-invariant metric for evaluating trajectory similarity. To validate the effectiveness of our contributions, we conduct comparative analyses against existing methods using a publicly available dataset. Moreover, we train StableNODEs to generate trajectories from RGB images for a picking task, achieving a success rate of 91.27%. Overall, StableNODEs present a modular framework adaptable to various problem domains by adjusting the training loss, attractor specifications, or the type of output mappings. Potential future expansions of our approach include mitigating unwanted critical points in Lyapunov functions and extending the capability to learn discontinuous DS for generating trajectories with contacts [26].

Appendix

A. Proof of Theorem 1

We split the analysis in three distinct cases:

- If $L(\boldsymbol{x}) \leq 0$ function $V_{\boldsymbol{\theta}}$ exponentially decreases since:

$$\begin{aligned} \nabla V_{\boldsymbol{\theta}}(\boldsymbol{x})\hat{\boldsymbol{f}}(\boldsymbol{x}) \leq -\alpha V_{\boldsymbol{\theta}}(\boldsymbol{x}) \implies \\ \dot{V}_{\boldsymbol{\theta}}(\boldsymbol{x}) \leq -\alpha V_{\boldsymbol{\theta}}(\boldsymbol{x}) \implies \\ V_{\boldsymbol{\theta}}(\boldsymbol{x}) \leq |V_{\boldsymbol{\theta}}(0)|e^{-\alpha t} \end{aligned}$$

- If $0 < L(\boldsymbol{x}) < \frac{1}{s}$ then the derivative of the Lyapunov function is :

$$\begin{aligned} \dot{V}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \nabla V_{\boldsymbol{\theta}}^T(\boldsymbol{x})\hat{\boldsymbol{f}}(\boldsymbol{x}) \implies \\ \dot{V}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \left(1 - \frac{\epsilon s L(\boldsymbol{x})}{|\nabla V_{\boldsymbol{\theta}}(\boldsymbol{x})|^2 + \epsilon}\right) \nabla V_\theta^T(\boldsymbol{x}) \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) \\ - \left(1 - \frac{\epsilon}{|\nabla V_{\boldsymbol{\theta}}(\boldsymbol{x})|^2 + \epsilon}\right) L(\boldsymbol{x}) \end{aligned}$$

After expanding $L(\boldsymbol{x}) = V_\theta^T(\boldsymbol{x})\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) + \alpha V_{\boldsymbol{\theta}}(\boldsymbol{x})$ and simplifying:

$$\begin{aligned} \dot{V}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{\epsilon - \epsilon s L(\boldsymbol{x})}{|\nabla V_{\boldsymbol{\theta}}(\boldsymbol{x})|^2 + \epsilon} \nabla V_\theta^T(\boldsymbol{x}) \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) \\ - \frac{|\nabla V_{\boldsymbol{\theta}}(\boldsymbol{x})|^2}{|\nabla V_{\boldsymbol{\theta}}(\boldsymbol{x})|^2 + \epsilon} \alpha V_{\boldsymbol{\theta}}(\boldsymbol{x}) \end{aligned}$$

Since $-\frac{|\nabla V_{\boldsymbol{\theta}}(\boldsymbol{x})|^2}{|\nabla V_{\boldsymbol{\theta}}(\boldsymbol{x})|^2+\epsilon}\alpha V_{\boldsymbol{\theta}}(\boldsymbol{x}) \leq 0$:

$$\dot{V}_{\boldsymbol{\theta}}(\boldsymbol{x}) \leq \frac{\epsilon - \epsilon s L(\boldsymbol{x})}{|\nabla V_{\boldsymbol{\theta}}(\boldsymbol{x})|^2 + \epsilon} \nabla V_{\theta}^T(\boldsymbol{x}) \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x})$$

Moreover, owing to the fact that $L(\boldsymbol{x}) < \frac{1}{s}$ it holds that $1 - sL(\boldsymbol{x}) > 0$ and $\nabla V_{\theta}^T(\boldsymbol{x}) \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) < \frac{1}{s} - \alpha V_{\boldsymbol{\theta}}(\boldsymbol{x})$. After substituting these expressions in the inequality for $\dot{V}_{\boldsymbol{\theta}}(\boldsymbol{x})$, we get:

$$\dot{V}_{\boldsymbol{\theta}}(\boldsymbol{x}) < \frac{\epsilon - \epsilon s L(\boldsymbol{x})}{|\nabla V_{\boldsymbol{\theta}}(\boldsymbol{x})|^2 + \epsilon} \left(\frac{1}{s} - \alpha V_{\boldsymbol{\theta}}(\boldsymbol{x}) \right)$$

The derivative of the energy function is rendered negative when $V_{\boldsymbol{\theta}}(\boldsymbol{x}) > \frac{1}{s\alpha}$. This implies that in the regions of the phase space where $0 < L(\boldsymbol{x}) < \frac{1}{s}$, $V_{\boldsymbol{\theta}}$ is decreasing until it reaches the set $\{\boldsymbol{x} \in X | V_{\boldsymbol{\theta}}(\boldsymbol{x}) \leq \frac{1}{s\alpha}\}$. Although the level sets of $V_{\boldsymbol{\theta}}$ after the training are not known a priori, the set $\{\boldsymbol{x} \in X | V_{\boldsymbol{\theta}}(\boldsymbol{x}) \leq \frac{1}{s\alpha}\}$ can be made arbitrary small by increasing s.

- If $L(\boldsymbol{x}) \geq \frac{1}{s}$ then the derivative of the candidate Lyapunov function is:

$$\begin{aligned} \dot{V}_{\boldsymbol{\theta}}(\boldsymbol{x}) = &\left(1 - \frac{\epsilon}{|\nabla V_{\boldsymbol{\theta}}(\boldsymbol{x})|^2 + \epsilon}\right) \nabla V_{\boldsymbol{\theta}}^T(\boldsymbol{x}) f_{\theta}(\boldsymbol{x}) \\ &- |\nabla V_{\boldsymbol{\theta}}(\boldsymbol{x})|^2 \frac{\dot{V}_{\boldsymbol{\theta}}(\boldsymbol{x})}{|\nabla V_{\boldsymbol{\theta}}(\boldsymbol{x})|^2 + \epsilon} \implies \\ \dot{V}_{\boldsymbol{\theta}}(\boldsymbol{x}) = &-\alpha \left(1 - \frac{\epsilon}{|\nabla V_{\boldsymbol{\theta}}(\boldsymbol{x})|^2 + \epsilon}\right) V_{\boldsymbol{\theta}}(\boldsymbol{x}) \end{aligned}$$

If $V_{\boldsymbol{\theta}}$ is a convex function with respect to x then if $L(\boldsymbol{x}) \geq \frac{1}{s} \implies \left(1 - \frac{\epsilon}{|\nabla V_{\boldsymbol{\theta}}(\boldsymbol{x})|^2+\epsilon}\right) > 0$, since $|\nabla V_{\boldsymbol{\theta}}(\boldsymbol{x})|^2 > 0$. That means that $V_{\boldsymbol{\theta}}$ is decreasing exponentially, except in the regions where $0 < L(\boldsymbol{x}) < \frac{1}{s}$. If $\epsilon = 0$ then the same analysis as [10] is followed.

If $\boldsymbol{X}$ is a closed and bounded set, then if $V_{\boldsymbol{\theta}}$ is made maximal on the boundary $\partial \boldsymbol{X}$, then it is guaranteed that $\boldsymbol{X}$ is forward invariant if $\boldsymbol{x}_0 \in \boldsymbol{X}$. This is because $V_{\boldsymbol{\theta}}$ is decreasing which means that $\boldsymbol{x}$ never reaches $\partial \boldsymbol{X}$.

If $\boldsymbol{X}_c$ is not empty then the strictest form the last equality can get is $\dot{V}_{\theta}(\boldsymbol{x}) = 0$, which would imply that the DS will be trapped in $\boldsymbol{X}_c$. Eventually, $\boldsymbol{x}$ will be asymptotically confined within the set $\boldsymbol{E}$. Note that avoiding critical points would require either special classes of energy functions [20] or a discontinuous DS [27]. Although entrapment in critical points is not optimal, our method accounts for it by avoiding the explosion of $\hat{\boldsymbol{f}}$ to infinity when they are encountered.

REFERENCES

[1] S. Calinon, "Learning from demonstration (programming by demonstration)," *Encyclopedia of robotics*, pp. 1–8, 2018.

[2] A. Fishman, C. Paxton, W. Yang, D. Fox, B. Boots, and N. Ratliff, "Collaborative interaction models for optimized human-robot teamwork," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 11 221–11 228.

[3] A. Sochopoulos, T. Poliero, D. Caldwell, J. Ortiz, and C. Di Natali, "Human-in-the-loop optimization of active back-support exoskeleton assistance via lumbosacral joint torque estimation," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 6090–6096.

[4] S. Schaal, "Is imitation learning the route to humanoid robots?" *Trends in cognitive sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[5] S. Manschitz, M. Gienger, J. Kober, and J. Peters, "Mixture of attractors: A novel movement primitive representation for learning motor skills from demonstrations," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 926–933, 2018.

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

[7] ——, "Learning control lyapunov function to ensure stability of dynamical system-based robot reaching motions," *Robotics and Autonomous Systems*, vol. 62, no. 6, pp. 752–765, 2014.

[8] M. A. Rana, A. Li, D. Fox, B. Boots, F. Ramos, and N. Ratliff, "Euclideanizing flows: Diffeomorphic reduction for learning stable dynamical systems," in *Learning for Dynamics and Control*. PMLR, 2020, pp. 630–639.

[9] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: learning attractor models for motor behaviors," *Neural computation*, vol. 25, no. 2, pp. 328–373, 2013.

[10] J. Z. Kolter and G. Manek, "Learning stable deep dynamics models," *Advances in neural information processing systems*, vol. 32, 2019.

[11] K. Neumann and J. J. Steil, "Learning robot motions with stable dynamical systems under diffeomorphic transformations," *Robotics and Autonomous Systems*, vol. 70, pp. 1–15, 2015.

[12] J. Zhang, H. B. Mohammadi, and L. Rozo, "Learning riemannian stable dynamical systems via diffeomorphisms," in *6th Annual Conference on Robot Learning*, 2022.

[13] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," *Advances in neural information processing systems*, vol. 31, 2018.

[14] K. Miao and K. Gatsis, "Learning robust state observers using neural odes," in *Learning for Dynamics and Control Conference*. PMLR, 2023, pp. 208–219.

[15] L. Böttcher, N. Antulov-Fantulin, and T. Asikis, "Ai pontryagin or how artificial neural networks learn to control dynamical systems," *Nature communications*, vol. 13, no. 1, p. 333, 2022.

[16] L. Böttcher and T. Asikis, "Near-optimal control of dynamical systems with neural ordinary differential equations," *Machine Learning: Science and Technology*, vol. 3, no. 4, p. 045004, 2022.

[17] A. Toshev and C. Szegedy, "Deeppose: Human pose estimation via deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1653–1660.

[18] I. D. J. Rodriguez, A. Ames, and Y. Yue, "Lyanet: A lyapunov framework for training neural odes," in *International Conference on Machine Learning*. PMLR, 2022, pp. 18 687–18 703.

[19] S. M. Richards, F. Berkenkamp, and A. Krause, "The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems," in *Conference on Robot Learning*. PMLR, 2018, pp. 466–476.

[20] D. E. Koditschek and E. Rimon, "Robot navigation functions on manifolds with boundary," *Advances in applied mathematics*, vol. 11, no. 4, pp. 412–442, 1990.

[21] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," *arXiv preprint arXiv:1605.08803*, 2016.

[22] O. Schutze, X. Esquivel, A. Lara, and C. A. C. Coello, "Using the averaged hausdorff distance as a performance measure in evolutionary multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 4, pp. 504–522, 2012.

[23] P. Kidger, "On neural differential equations," *arXiv preprint arXiv:2202.02435*, 2022.

[24] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, and Y. Zhu, "robosuite: A modular simulation framework and benchmark for robot learning," in *arXiv preprint arXiv:2009.12293*, 2020.

[25] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín, "What matters in learning from offline human demonstrations for robot manipulation," in *arXiv preprint arXiv:2108.03298*, 2021.

[26] B. Bianchini, M. Halm, and M. Posa, "Simultaneous learning of contact and continuous dynamics," in *Conference on Robot Learning*. PMLR, 2023, pp. 3966–3978.

[27] J. Cortes, "Discontinuous dynamical systems," *IEEE Control Systems Magazine*, vol. 28, no. 3, pp. 36–73, 2008.