Stable Motion Primitives via Imitation and Contrastive Learning

Pérez-Dattari, Rodrigo; Kober, Jens

DOI
10.1109/TRO.2023.3289597

Publication date
2023

Document Version
Final published version

Published in
IEEE Transactions on Robotics

Citation (APA)
Pérez-Dattari, R., & Kober, J. (2023). Stable Motion Primitives via Imitation and Contrastive Learning. *IEEE Transactions on Robotics*, *39*(5), 3909-3928. https://doi.org/10.1109/TRO.2023.3289597

Stable Motion Primitives via Imitation and Contrastive Learning

Rodrigo Pérez-Dattari, *Member, IEEE*, and Jens Kober, *Senior Member, IEEE*

***Abstract*—Learning from humans allows nonexperts to program robots with ease, lowering the resources required to build complex robotic solutions. Nevertheless, such data-driven approaches often lack the ability to provide guarantees regarding their learned behaviors, which is critical for avoiding failures and/or accidents. In this work, we focus on reaching/point-to-point motions, where robots must always reach their goal, independently of their initial state. This can be achieved by modeling motions as dynamical systems and ensuring that they are globally asymptotically stable. Hence, we introduce a novel Contrastive Learning loss for training deep neural networks (DNN) that, when used together with an Imitation Learning loss, enforces the aforementioned stability in the learned motions. Differently from previous work, our method does not restrict the structure of its function approximator, enabling its use with arbitrary DNNs and allowing it to learn complex motions with high accuracy. We validate it using datasets and a real robot. In the former case, motions are two- and four-dimensional, modeled as first- and second-order dynamical systems. In the latter, motions are three, four, and six-dimensional, of first and second order, and are used to control a 7-DoF robot manipulator in its end effector space and joint space.**

***Index Terms*—Contrastive learning, deep neural networks (DNNs), dynamical systems, imitation learning (IL), motion primitives.**

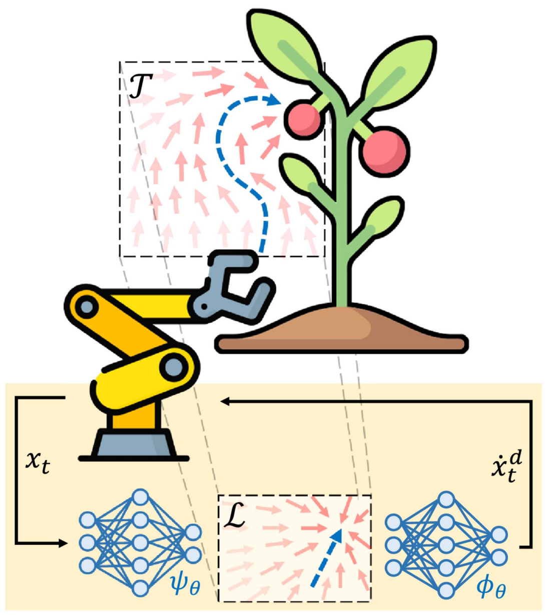

Fig. 1. Overview of a motion model learned using the proposed framework. The blue trajectory in the task space $\mathcal{T}$, shows the movement of the robot's end effector when starting from its current state x_t. The evolution of this trajectory is defined by the dynamical system $\dot{x}_t^d = \phi_\theta(\psi_\theta(x_t))$, which is represented with a vector field of red arrows in the rest of the space. Using Contrastive Learning, this system is coupled with a well-understood and stable dynamical system in the latent space $\mathcal{L}$ that ensures its stability.

I. Introduction

Imitation learning (IL) provides a framework that is intuitive for humans to use, without requiring them to be robotics experts. It allows robots to be programmed by employing methods similar to the ones humans use to learn from each other, such as demonstrations, corrections, and evaluations. This significantly reduces the resources needed for building robotic systems, making it particularly appealing for real-world applications (e.g., Fig. 1).

Nevertheless, due to their data-driven nature, IL methods often lack guarantees, such as ensuring that a robot's motion always reaches its target, independently of its initial state (e.g., Fig. 2). This can be a major limitation for implementing methods in the real world since it can lead to failures and/or accidents. To tackle these challenges, we can model motions as dynamical systems whose evolution describes a set of human demonstrations [1], [2], [3]. This is advantageous because 1) the model depends on the robot's state and it is learned offline, enabling the robot to adapt to changes in the environment during task execution, and 2) dynamical systems theory can be employed to analyze the behavior of the motion and provide guarantees.

In this work, we focus on learning dynamical systems from demonstrations to model *reaching motions*, as a wide range of tasks requires robots to reach goals, e.g., hanging objects, pick-and-place of products, crop harvesting, and button pressing. Furthermore, these motions can be sequenced to model cyclical behaviors, extending their use for such problems as well [4].

A reaching motion modeled as a dynamical system is considered to be globally asymptotically stable if the robot always reaches its goal, independently of its initial state. In this work,

Manuscript received 20 February 2023; revised 20 May 2023; accepted 5 June 2023. Date of publication 10 August 2023; date of current version 4 October 2023. This work was supported by the Netherlands Organization for Scientific Research Project Cognitive Robots for Flexible Agro-Food Technology under Grant P17-01. This paper was recommended for publication by Associate Editor J. Bohg and Editor W. Burgard upon evaluation of the reviewers' comments. *(Corresponding author: Rodrigo Pérez-Dattari.)*

The authors are with the Cognitive Robotics Department, Delft University of Technology, 2628 CD Delft, Netherlands (e-mail: r.j.perezdattari@tudelft.nl; j.kober@tudelft.nl).

More details regarding the real-world experiments are presented in: https://youtu.be/OM-2edHBRfc.

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TRO.2023.3289597.

Digital Object Identifier 10.1109/TRO.2023.3289597

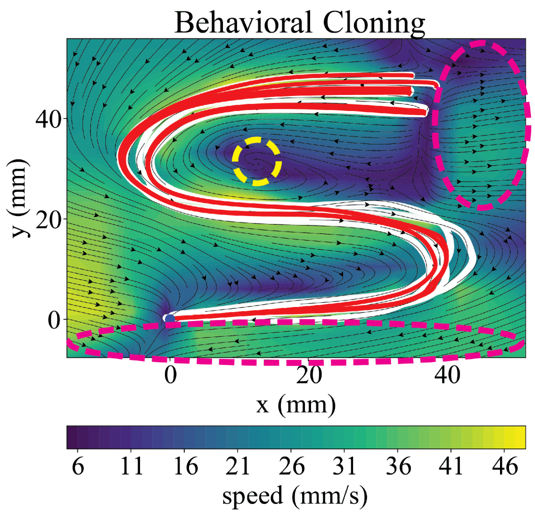

Fig. 2. Example of a motion learned using behavioral cloning (BC). White curves represent demonstrations. Red curves represent learned motions when starting from the same initial positions as the demonstrations. The arrows indicate the vector field of the learned dynamical system. The yellow dotted line shows a region with a spurious attractor. The magenta lines show regions where the trajectories diverge away from the goal.

we will refer to such systems as *stable* for short. Notably, by employing dynamical systems theory, stability in reaching motions can be enforced when learning from demonstrations, providing guarantees to these learning frameworks.

In the literature, there is a family of works that use this approach to learn stable motions from demonstrations [2], [3], [5]. However, these often constrain the structure of their learning models to meet certain conditions needed to guarantee the stability of their motions, e.g., by enforcing the learning functions to be invertible [3], [6], [7] or positive/negative definite [2], [8], [9]. Although these constraints ensure stability, they limit the applicability of the methods to a narrow range of models. For example, if a novel promising deep neural network (DNN) architecture is introduced in the literature, it would not be straightforward/possible to use it in such frameworks, since, commonly, DNN architectures do not have this type of constraints. Furthermore, the learning flexibility of a function approximator is limited if its structure is restricted, which can hurt its accuracy performance when learning motions.

Hence, in the context of DNNs,[1] we propose a novel method for learning stable motions without constraining the structure of the function approximator. To achieve this, we introduce a contrastive loss [11], [12] to enforce stability in dynamical systems modeled with arbitrary DNN architectures. This is achieved by transferring the stability properties of a simple, stable dynamical system to the more complex system that models the demonstrations (see Fig. 1). To the best of authors' knowledge, this is the first approach that learns to generate stable motions with DNNs without relying on a specific architecture type.

[1]We understand DNNs as a collection of machine learning algorithms that learn in a hierarchical manner, i.e., the function approximator consists of a composition of multiple functions, and are optimized by means of backpropagation [10].

We validate our method using both simulated and real-world experiments, demonstrating its ability to successfully scale in terms of the order and dimensionality of the dynamical system. Furthermore, we showcase its capabilities for controlling a 7-DoF robotic manipulator in both joint and end-effector space. Lastly, we explore potential extensions for the method, such as combining motions by learning multiple systems within a single DNN architecture.

The rest of this article is organized as follows. Related works are presented in Section II. Section III describes the background and problem formulation of our method. Section IV develops the theory required to introduce the contrastive loss, introduces it, and explains how we employ it in the context of IL. Experiments and results are divided into Sections V, VI, and VII. Section V validates our method using datasets of motions modeled as first-order and second-order dynamical systems. These motions are learned from real data, but they are evaluated without employing a real system. Section VI validates the method in a real robot, and Section VII studies possible ways of extending it. Finally, Section VIII concludes this article.

II. Related Works

Several works have approached the problem of learning motions modeled as dynamical systems from demonstrations while ensuring their stability. By observing if these works employ either *time-varying* or *time-invariant* dynamical systems, we can divide them into two groups. In time-varying dynamical systems, the evolution of the system explicitly depends on time (or a phase). In contrast, time-invariant dynamical systems do not depend on time *directly*, but only through its time-varying input (i.e., the state of the system). The property of a system being either time-invariant or not, conditions the type of strategies that can be employed to enforce its stability. Hence, for this work, it makes sense to make a distinction between these systems.

One seminal work of IL that addresses stability for time-varying dynamical systems introduces dynamical movement primitives [1]. This method takes advantage of the time-dependency (via the phase of the *canonical system*) of the dynamical system to enforce its nonlinear part, which captures the behavior of the demonstration, to vanish as time goes to infinity. Then, they build the remainder of the system to be a function that is well-understood and stable by construction. Hence, since the nonlinear part of the motion will eventually vanish, its stability can be guaranteed. In the literature, some works extend this idea with probabilistic formulations [13], [14], and others have extended its use to the context of DNNs [15], [16], [17].

These time-varying dynamical system approaches are well-suited for when the target motions have clear temporal dependencies. However, they can generate undesired behaviors when encountering perturbations (assuming the time/phase is not explicitly modulated), and they lack the ability to model different behaviors for different regions of the robot's state space. In contrast, time-invariant dynamical systems can easily address these shortcomings, but they can be more challenging to employ when motions contain strong temporal dependencies.

Therefore, IL formulations with such systems are considered to be complementary to the ones that employ time-varying systems [18], [19]. In this work, we focus on time-invariant dynamical systems.

An important family of works has addressed the problem of modeling stable motions as time-invariant dynamical systems. These approaches often constrain the structure of the dynamical systems to ensure Lyapunov stability by design. In this context, one seminal work introduces the stable estimator of dynamical systems (SEDS) [2]. This approach imposes constraints on the structure of Gaussian mixture regressions (GMR), ensuring stability in the generated motions.

Later, this idea inspired other works to explicitly learn Lyapunov functions that are consistent with the demonstrations and *correct* the transitions of the learned dynamical system such that they are stable according to the learned Lyapunov function [8], [9], [20]. Furthermore, several extensions of SEDS have been proposed, for instance, by using physically consistent priors [21], contraction theory [22], or diffeomorphisms [23].

Moreover, some of these ideas, such as the use of contraction theory or diffeomorphisms have also been used outside the scope of SEDS. Contraction theory ensures stability by enforcing the distance between the trajectories of a system to reduce, according to a given metric, as the system evolves. Hence, it has been employed to learn stable motions from demonstrations [24], [25]. In contrast, diffeomorphisms can be employed to transfer the stability properties of a stable and well-understood system, to a complex nonlinear system that models the behavior of the demonstrations. Hence, this strategy has also been employed to learn stable motions from demonstrations [3], [6], [7]. As we explain in Section IV-B1, our method is closely related to these approaches. It is worth noting that of the mentioned strategies, only [7] models stable stochastic dynamics. However, this concept could also be explored with other encoder–decoder stochastic models, e.g., [13].

Understandably, all of these methods constrain some part of their learning framework to ensure stability. From one point of view, this is advantageous, since they can guarantee stability. However, in many cases, this comes with the cost of reducing the flexibility of the learned motions (i.e., loss in accuracy). Notably, some recent methods have managed to reduce this loss in accuracy [3], [7]; however, they are still limited in terms of the family of models that can be used with these frameworks, which harms their scalability. Consequently, in this work, we address these limitations by enforcing the stability of the learned motions as a soft constraint and showing its effectiveness in obtaining stable, accurate, and scalable motions.

III. Preliminaries

A. Dynamical Systems as Movement Primitives

In this work, we model motions as nonlinear time-invariant dynamical systems defined by the equation

$$\dot{x} = f(x)$$

where $x \in \mathbb{R}^n$ is the system's state and $f : \mathbb{R}^n \to \mathbb{R}^n$ is a nonlinear continuous and continuously differentiable function. The evolution defined by this dynamical system is transferred to the robot's state by tracking it with a lower level controller.

B. Global Asymptotic Stability

We are interested in solving reaching tasks. From a dynamical system perspective, this means that we want to construct a system where the goal state $x_g \in \mathbb{R}^n$ is a *globally asymptotically stable* equilibrium. An equilibrium x_g is globally asymptotically stable if $\forall x \in \mathbb{R}^n$

$$\lim_{t\to\infty} ||x - x_g|| = 0.$$

Note that for this condition to be true, the time derivative of the dynamical system at the attractor must be zero, i.e., $\dot{x} = f(x_g) = 0$.

For simplicity, we use the word *stable* to refer to these systems.

C. Problem Formulation

Consider the scenario where a robot aims to learn a reaching motion, in a given space $\mathcal{T} \subset \mathbb{R}^n$ and with respect to a given goal $x_g \in \mathcal{T}$, based on a set of demonstrations $\mathcal{D}$. The robot is expected to imitate the behavior shown in the demonstrations while always reaching x_g, regardless of its initial state.

The dataset $\mathcal{D}$ contains N demonstrations in the form of trajectories τ_i, such that $\mathcal{D} = (\tau_0, \tau_1, \ldots, \tau_{N-1})$. Each one of these trajectories contains the evolution of a dynamical system with discrete-time states $x_t \in \mathcal{T}$ when starting from an initial state x_0 and it transitions for T time steps t of size Δt. Hence, $\tau_i = (x_0^i, x_1^i, \ldots, x_{T-1}^i)$, where T does not have to be the same for every demonstration, and here we added the superscript i to the states to explicitly indicate that they belong to the trajectory τ_i. Note, however, that the state superscript will not be used for the rest of this article

We assume that these trajectories are drawn from the distribution $p^*(\tau)$, where every transition belonging to a trajectory sampled from this distribution follows the *optimal* (according to the demonstrator's judgment) dynamical system f^*. On the other hand, the robot's motion is modeled as the parametrized dynamical system $f_\theta^\mathcal{T}$, which induces the trajectory distribution $p_\theta(\tau)$, where θ is the parameter vector.

Then, the objective is to find θ^* such that the distance between the trajectory distributions induced by the human and learned dynamical system is minimized while ensuring the stability of the motions generated with $f_\theta^\mathcal{T}$ toward x_g. This can be formulated as the minimization of the (forward) Kullback–Leibler divergence between these distributions [26], subject to a stability constraint of the learned system

$$\begin{aligned}
&\theta^* = \arg\min_\theta \quad D_{\mathrm{KL}}\left(p^*(\tau)||p_\theta(\tau)\right)\\
&\text{s.t.} \lim_{t\to\infty} ||x_t - x_g|| = 0,\\
&\forall x_t \in \mathcal{T} \text{ evolving with } f_\theta^\mathcal{T}.
\end{aligned}$$

IV. Methodology

We aim to learn motions from demonstrations modeled as nonlinear time-invariant dynamical systems. In this context, we present the *CONvergent Dynamics from demOnstRations* (CONDOR) framework. This framework learns the parametrized function $f_\theta^{\mathcal{T}}$ using human demonstrations and ensures that this dynamical system has a globally asymptotically stable equilibrium at x_g while being accurate w.r.t. the demonstrations.

To achieve this, we extend the IL problem with a novel loss ℓ_{stable} based on Contrastive Learning (CL) [11] that aims to ensure the stability of the learned system. Hence, if the IL problem minimizes the loss ℓ_{IL}, our framework minimizes

$$\ell_{\text{CIL}} = \ell_{\text{IL}} + \lambda \ell_{\text{stable}}$$

where $\lambda \in \mathbb{R}$ is a weight. We refer to ℓ_{CIL} as the *contrastive imitation learning* (CIL) loss.

A. Structure of CONDOR

The objective of ℓ_{stable} is to ensure that $f_\theta^{\mathcal{T}}$ shares stability properties with a simple and well-understood system. We will refer to this system as $f^{\mathcal{L}}$, which is designed to be stable by construction. Consequently, if $f^{\mathcal{L}}$ is stable, then $f_\theta^{\mathcal{T}}$ will also be stable.

Since $f_\theta^{\mathcal{T}}$ is parametrized by a DNN, we can define $f^{\mathcal{L}}$ to reside within the output of one of the hidden layers of $f_\theta^{\mathcal{T}}$. This formulation might seem arbitrary; however, it will be shown later that it enables us to introduce the *stability conditions*, which serve as the foundation for designing ℓ_{stable}. Therefore, we define the dynamical system $f_\theta^{\mathcal{T}}$ as a composition of two functions, ψ_θ and ϕ_θ

$$\dot{x}_t = f_\theta^{\mathcal{T}}(x_t) = \phi_\theta(\psi_\theta(x_t)).$$

$\forall x_t \in \mathcal{T}$. Note that $f_\theta^{\mathcal{T}}$ is a standard DNN with L layers. ψ_θ denotes layers $1 \ldots l$, and ϕ_θ layers $l+1 \ldots L$. We define the output of layer l as the latent space $\mathcal{L} \subset \mathbb{R}^n$. Moreover, for simplicity, although we use the same θ notation for both ψ_θ and ϕ_θ, each symbol actually refers to a different subset of parameters within θ. These subsets together form the full parameter set in $f_\theta^{\mathcal{T}}$.

Then, the dynamical system $f^{\mathcal{L}}$ is defined to evolve within $\mathcal{L}$.[2] This system is constructed to be stable at the equilibrium $y_g = \psi(x_g)$, and can be described by

$$\dot{y}_t^{\mathcal{L}} = f^{\mathcal{L}}(y_t^{\mathcal{L}}).$$

$\forall y_t^{\mathcal{L}} \in \mathcal{L}$. Here, $y_t^{\mathcal{L}}$ corresponds to the *latent state variables* that evolve according to $f^{\mathcal{L}}$.

Lastly, it is necessary to introduce a third dynamical system. This system represents the evolution in $\mathcal{L}$ of the states visited by $f_\theta^{\mathcal{T}}$ when mapped using ψ_θ, which yields the relationship

$$\dot{y}_t^{\mathcal{T}} = f_\theta^{\mathcal{T}\to\mathcal{L}}(x_t) = \frac{\partial \psi_\theta(x_t)}{\partial t}.$$

$\forall y_t^{\mathcal{T}} \in \mathcal{L}$, where $y_t^{\mathcal{T}}$ corresponds to the latent variables that evolve according to $f_\theta^{\mathcal{T}\to\mathcal{L}}$.

Fig. 3 summarizes the introduced dynamical systems.

[2]Note that before training, this system will not completely reside in $\mathcal{L}$, since it is allowed us to evolve outside the image of ψ_θ.

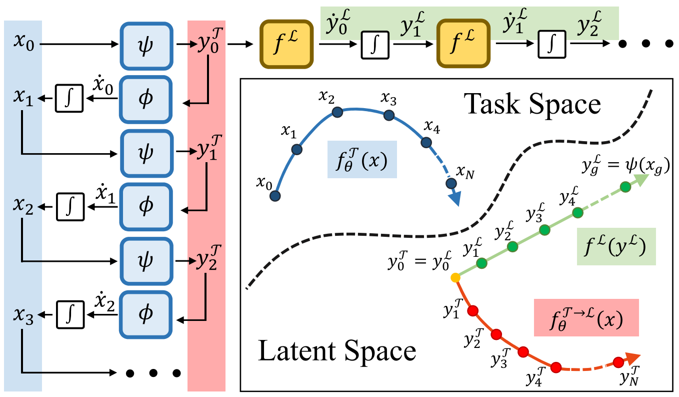

Fig. 3. Structure of CONDOR. We show an example of discrete-time trajectories generated with $f_\theta^{\mathcal{T}}$, $f^{\mathcal{L}}$ and $f_\theta^{\mathcal{T}\to\mathcal{L}}$ before training the DNN. Starting from an initial point x_0, a trajectory is generated in $\mathcal{T}$ using $f_\theta^{\mathcal{T}}$ (blue) and two trajectories are generated in $\mathcal{L}$. One of them follows $f_\theta^{\mathcal{T}\to\mathcal{L}}$ (red), and the other follows $f^{\mathcal{L}}$ (green).

B. Stability Conditions

The above-presented dynamical systems allow us to introduce the stability conditions. These conditions state that if $f_\theta^{\mathcal{T}\to\mathcal{L}}$ exhibits the same behavior as $f^{\mathcal{L}}$, and only x_g maps to $\psi_\theta(x_g)$, then $f_\theta^{\mathcal{T}}$ is stable. We formally introduce them as follows:

Theorem 1 (Stability conditions): Let $f_\theta^{\mathcal{T}}$, $f_\theta^{\mathcal{T}\to\mathcal{L}}$ and $f^{\mathcal{L}}$ be the dynamical systems introduced in Section IV-A. Then, x_g is a globally asymptotically stable equilibrium of $f_\theta^{\mathcal{T}}$ if, $\forall x_t \in \mathcal{T}$:

1) $f_\theta^{\mathcal{T}\to\mathcal{L}}(x_t) = f^{\mathcal{L}}(y_t^{\mathcal{T}})$;
2) $\psi_\theta(x_t) = y_g \Rightarrow x_t = x_g$.

Proof: Since $f^{\mathcal{L}}$ is globally asymptotically stable at y_g, condition 1) indicates that as $t \to \infty$, $y_t^{\mathcal{T}} = y_t^{\mathcal{L}} \to y_g$. However, from condition 2) we know that $y_t^{\mathcal{T}} = \psi_\theta(x_t) = y_g$ is only possible if $x_t = x_g$. Hence, as $t \to \infty$, $x_t \to x_g$. Then, x_g is globally asymptotically stable in $f_\theta^{\mathcal{T}}(x_t)$. $\square$

Consequently, we aim to design ℓ_{stable} such that it enforces the stability conditions in the presented dynamical systems by optimizing ψ_θ and ϕ_θ.

1) Connection With Diffeomorphism-Based Methods: It is interesting to note that the stability conditions make ψ_θ converge to a *diffeomorphism* between $\mathcal{T}$ and $\mathcal{L}$ (proof in Appendix A). Consequently, our approach becomes tightly connected to methods that ensure stability using diffeomorphic function approximators [3], [6], [7]. However, differently from these methods, we do not require to take into account the structure of the function approximator and explicit relationships between $f_\theta^{\mathcal{T}}$ and $f_\theta^{\mathcal{T}\to\mathcal{L}}$.

C. Enforcing Stability

In this section, we introduce a method that enforces the stability conditions in $f_\theta^{\mathcal{T}}$.

1) First Condition ($f_\theta^{\mathcal{T}\to\mathcal{L}} = f^{\mathcal{L}}$): The first stability condition can be enforced by minimizing the distance between the states visited by the dynamical systems $f_\theta^{\mathcal{T}\to\mathcal{L}}$ and $f^{\mathcal{L}}$ when starting from the same initial condition. Hence, $\forall y_t^{\mathcal{T}}, y_t^{\mathcal{L}} \in \mathcal{L}$ a loss can be defined as $\ell_{\text{match}} = d(y_t^{\mathcal{T}}, y_t^{\mathcal{L}})$, where $d(\cdot,\cdot)$ is a distance function.

2) Second Condition ($\psi_\theta(x_t) = y_g \Rightarrow x_t = x_g$): The second stability condition, however, can be more challenging to obtain, since we do not have a direct way of optimizing this in a DNN. Therefore, to achieve this, we introduce the following proposition.

Proposition 1 (Surrogate stability conditions): The second stability condition of Theorem 1, i.e., $\psi_\theta(x_t) = y_g \Rightarrow x_t = x_g$, $\forall x_t \in \mathcal{T}$, is true if:

1) $f_\theta^{\mathcal{T}\to\mathcal{L}}(x_t) = f^{\mathcal{L}}(y_t^{\mathcal{T}}), \forall x_t \in \mathcal{T}$ (stability condition 1);
2) $y_{t-1}^{\mathcal{T}} \neq y_t^{\mathcal{T}}, \forall x_t \in \mathcal{T} \setminus \{x_g\}$.

Proof: If $y_{t-1}^{\mathcal{T}} = \psi(x_{t-1}) = y_g$ the first condition implies that $y_g = y_{t-1}^{\mathcal{T}} = y_t^{\mathcal{T}}$, since $f^{\mathcal{L}}(y_g) = 0$. Consequently, given the second condition $y_{t-1}^{\mathcal{T}} \neq y_t^{\mathcal{T}}, \forall x_t \neq x_g$, this is only possible if $x_{t-1} = x_g$.

In other words, Proposition 1 indicates that if the first stability condition is true; then, we can obtain its second condition by enforcing $y_{t-1}^{\mathcal{T}} \neq y_t^{\mathcal{T}}, \forall x_t \neq x_g$. Notably, by enforcing this, the stability conditions are also enforced. Consequently, we refer to the conditions of Proposition 1 as the *surrogate stability conditions*.

Then, it only remains to define a loss ℓ_{sep} that enforces the second surrogate stability condition in $f_\theta^{\mathcal{T}}$. However, before doing so, note that the surrogate conditions aim to push some points together (i.e., $y_t^{\mathcal{T}}$ and $y_t^{\mathcal{L}}$) and separate others (i.e., $y_{t-1}^{\mathcal{T}}$ and $y_t^{\mathcal{T}}$). Hence, this problem overlaps with the CL and deep metric learning literature [12].

3) Contrastive Learning: The problem of pushing some points together (ℓ_{match}) and separating others (ℓ_{sep}), is equivalent to the problem that the pairwise contrastive loss, from the CL literature, optimizes [11]. This loss computes a cost that depends on positive and negative samples. Its objective is to reduce the distance between positive samples and separate negative samples beyond some margin value $m \in \mathbb{R}^+$.

In our problem, positive samples are defined as $y_t^{\mathcal{L}}$ and $y_t^{\mathcal{T}}$, and negative samples are defined as $y_{t-1}^{\mathcal{T}}$ and $y_t^{\mathcal{T}}$. The loss for positive samples is the same as ℓ_{match}. Differently, for negative samples, this method separates points by minimizing $\ell_{\text{sep}} = \max(0, m - d(y_{t-1}^{\mathcal{T}}, y_t^{\mathcal{T}})), \forall y_{t-1}^{\mathcal{T}}, y_t^{\mathcal{T}} \in \mathcal{L}$. If their distance is smaller than m, $m - d(y_{t-1}^{\mathcal{T}}, y_t^{\mathcal{T}}) > 0$, which is minimized until their distance is larger than m and $m - d(y_{t-1}^{\mathcal{T}}, y_t^{\mathcal{T}}) < 0$.

Commonly, the squared l^2-norm is used as the distance metric. Moreover, this loss is optimized along a trajectory starting at $t = 1$, which is a state sampled randomly from the task space $\mathcal{T}$. Then, we define a contrastive loss for motion stability as

$$\underbrace{\sum_{b=0}^{B^s-1}\sum_{t=1}^{H^s} \underbrace{||y_{t,b}^{\mathcal{L}} - y_{t,b}^{\mathcal{T}}||_2^2}_{\ell_{\text{match}}} + \underbrace{\max(0, m - ||y_{t,b}^{\mathcal{T}} - y_{t-1,b}^{\mathcal{T}}||_2)^2}_{\ell_{\text{sep}}}}_{\ell_{\text{stable}}} \tag{1}$$

where $B^s, H^s \in \mathbb{N}^+$ are the batch size corresponding to the number of samples used at each training iteration of the DNN and H^s is the trajectory length used for training, respectively.

Note that (1) does not take into account the fact that ℓ_{sep} should not be applied at y_g. However, in practice, it is very unlikely to sample x_g, so we do not deem it necessary to explicitly consider this case. Furthermore, the loss ℓ_{match} enforces $f^{\mathcal{T}}(y_g) = 0$, which also helps to keep $y_{t-1}^{\mathcal{T}}$ and $y_t^{\mathcal{T}}$ together when $y_{t-1}^{\mathcal{T}} = y_g$.

4) Relaxing the Problem: We can make use of the CL literature to use other losses to solve this problem. More specifically, we study the triplet loss [27] as an alternative to the pairwise loss. We call this version CONDOR (relaxed), since, in this case, the positive samples $y_t^{\mathcal{T}}$ and $y_t^{\mathcal{L}}$ are pushed closer, but it is not a requirement for them to be the same. Hence, we aim to observe if learning a specific structure in $\mathcal{L}$ is enough to enforce stability in $f_\theta^{\mathcal{T}}$, even though (1) is not solved exactly. This allows us to compare different features between losses, such as generalization capabilities.

D. Boundaries of the Dynamical System

We enforce the stability of a motion in the region $\mathcal{T}$ by randomly sampling points from it and minimizing (1). Since this property is learned by a DNN, stability cannot be ensured in regions of the state space where this loss is not minimized, i.e., outside of $\mathcal{T}$. Therefore, it is crucial to ensure that if a point belongs to $\mathcal{T}$, its evolution will not leave $\mathcal{T}$. In other words, $\mathcal{T}$ must be a *positively invariant set* w.r.t. $f_\theta^{\mathcal{T}}$ [9], [28].

To address this, we design the dynamical system such that, by construction, is not allowed us to leave $\mathcal{T}$. This can be easily achieved by projecting the transitions that leave $\mathcal{T}$ back to its boundary, i.e., if a point $x_t \in \mathcal{T}$ transitions to a point $x_{t+1} \notin \mathcal{T}$; then, it is projected to the boundary of $\mathcal{T}$. In this work, $\mathcal{T}$ is a hypercube; consequently, we apply an orthogonal projection by saturating/clipping the points that leave $\mathcal{T}$.

Note that this saturation is always applied, i.e., during the training and evaluation of the dynamical system. Hence, the stability conditions of Theorem 1 are imposed on a system that evolves in the positively invariant set $\mathcal{T}$.

E. Designing $f^{\mathcal{L}}$

So far, we presented a method for coupling two dynamical systems such that they share stability properties; however, we assumed that $f^{\mathcal{L}}$ existed. In reality, we must design this function such that it is stable by construction. Although several options are possible, in this work, we define $f^{\mathcal{L}}$ as

$$\dot{y}_t = \alpha \odot (y_g - y_t) \tag{2}$$

where $\alpha \in \mathbb{R}^n$ corresponds to the gains vector and $\odot$ to the element-wise/hadamard product. If $\alpha_i > 0$,[3] this system monotonically converges to y_g [29], where α_i corresponds to the ith element of α.

1) Adaptive Gains: In the simplest case, α is a fixed, predefined, value; however, the performance of the learned mappings ψ and ϕ is susceptible to the selected value of α. Alternatively, to provide more flexibility to the framework, we propose to define α as a trainable function that depends on the current latent

[3]This holds for the continuous-time case. However, we approximate the evolution of this system via the forward Euler integration method. Then, the system can be written for the discrete-time case as $y_{t+1} = Ay_t$, where $A = I + \text{diag}(-\alpha)\Delta t$, assuming $y_g = 0$ without loss of generality. To ensure stability, the absolute value of the eigenvalues of A must be less than one; then, $0 < \alpha_i < 2/\Delta t$.

state y_t, i.e., $\alpha(y_t)$. Then, the parameters of α can be optimized using the same losses employed to train ψ and ϕ, since it is connected to the rest of the network and the training error can be propagated through it. Note that this system is stable under the same condition for α as before, as shown in Appendix B.

F. BC of Dynamical Systems

Finally, we need to optimize an IL loss ℓ_{IL} such that the learned dynamical system $f_\theta^\mathcal{T}$ follows the demonstrations of the desired motion. For simplicity, we opt to solve a behavioral cloning (BC) problem; however, in principle, any other IL approach can be used. As described in Section III, this can be achieved by minimizing the (forward) Kullback–Leibler divergence between the demonstration's trajectory distribution and the trajectory distribution induced by the learned dynamical system. Note that this problem formulation is equivalent to applying maximum likelihood estimation (MLE) between these distributions [26]. Hence, we can rewrite it as

$$f_\theta^\mathcal{T} = \underset{f_\theta^\mathcal{T}\in\mathcal{F}}{\arg\max}\, \mathbb{E}_{\tau\sim p^*(\tau)}\left[\ln p_\theta(\tau)\right]. \tag{3}$$

If we note that $p_\theta(\tau)$ is a product of conditional transition distributions $p_\theta(x_{t+1}|x_t)$, we can rewrite it as $p_\theta(\tau) = \Pi_{t=0}^{T-1} p_\theta(x_{t+1}|x_t)p(x_0)$, where $p(x_0)$ is the initial state probability distribution. Replacing this in (3) and ignoring constants, we obtain

$$f_\theta^\mathcal{T} = \underset{f_\theta^\mathcal{T}\in\mathcal{F}}{\arg\max}\, \mathbb{E}_{\substack{x_{t+1}\sim p^*(x_{t+1}|x_t),\\ x_t\sim p^{t*}(x_t)}}\left[\sum_{t=0}^{T-1} \ln p_\theta(x_{t+1}|x_t)\right]$$

where $p^{t*}(x_t)$ is the probability distribution of states at time step t, and $p^*(x_{t+1}|x_t)$ is the distribution of transitioning to state x_{t+1} given that the system is in some state x_t. Both of these distributions are induced by the dynamical system f^*.

In practice, however, we do not have an analytical representation of the distributions $p^{t*}(x_t)$ and $p^*(x_{t+1}|x_t)$. Therefore, the problem has to be estimated through empirical evaluations of this objective, which is achieved using the demonstrations present in the dataset $\mathcal{D}$. Then, we can solve this problem iteratively [30] by randomly sampling batches of B^i trajectories from $\mathcal{D}$ at each iteration and maximizing

$$f_\theta^\mathcal{T} = \underset{f_\theta^\mathcal{T}\in\mathcal{F}}{\arg\max} \sum_{b=0}^{B^i-1}\sum_{t=0}^{T-1} \ln p_\theta(x^*_{t+1,b}|x_{t,b}) \tag{4}$$

where the subscript b has been added to the states indicating their correspondence to the different trajectories of B.

To solve this problem, we can assume the transition distribution of the learning system to be a Gaussian with fixed covariance, and a mean corresponding to the *forward Euler integration* [31] of $f_\theta^\mathcal{T}$ for the given state $x_{t,b}$, i.e., $x_{t+1,b} = x_{t,b} + f_\theta^\mathcal{T}(x_{t,b})\Delta t$, where Δt corresponds to the time step size. Furthermore, the same Gaussian assumption is made for the demonstration's distribution $p^*(x_{t+1}|x_t)$; however, since its transitions are obtained directly from the demonstrations, it is not necessary to integrate in this case. Then, (4) reduces to the mean squared error (MSE) minimization between the mean of

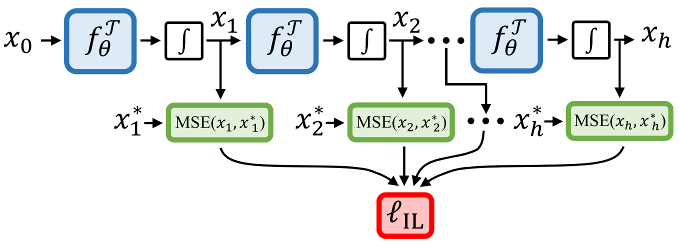

Fig. 4. Multistep IL loss for one sample when using backpropagation through time, where $h = H^i$.

the demonstration's distribution $p^*(x_{t+1}|x_t)$, and the mean of the learning distribution $p_\theta(x_{t+1}|x_t)$ [32], i.e.,

$$f_\theta^\mathcal{T} = \underset{f_\theta^\mathcal{T}\in\mathcal{F}}{\arg\min} \sum_{b=0}^{B^i-1}\sum_{t=0}^{T-1} ||x^*_{t+1,b} - \left(x_{t,b} + f_\theta^\mathcal{T}(x_{t,b})\Delta t\right)||_2^2.$$

In practice, however, if the trajectories of the demonstrations are too long, due to computation or complexity limitations, it might not be convenient to optimize this objective for the complete trajectories. Therefore, this problem can be simplified by allowing the initial conditions of the demonstration batches to be at any time step $t' \in \{0,\ldots,T-1\}$, and optimizing the problem for some time horizon $H^i \leq T$. Consequently, we get the loss ℓ_{IL} that we employ to solve the BC problem in this work:

$$f_\theta^\mathcal{T} = \underset{f_\theta^\mathcal{T}\in\mathcal{F}}{\arg\min} \underbrace{\sum_{b=0}^{B^i-1}\sum_{t=t'}^{H^i-1} ||x^*_{t+1,b} - x_{t,b} - f_\theta^\mathcal{T}(x_{t,b})\Delta t||_2^2}_{\ell_{\text{IL}}}. \tag{5}$$

G. Compounding Errors and Multistep Learning

Commonly, (5) is solved as a single-step prediction problem (i.e., $H^i = 1$) by computing only one transition from $x_{t',b}$ using $f_\theta^\mathcal{T}$ and comparing it against $x^*_{t'+1,b}$. Nevertheless, in practice, the learned dynamical system is applied recursively, i.e., assuming perfect tracking, every prediction is computed as a function of a previously computed output using the following equation:

$$x_h = x_{h-1} + f_\theta^\mathcal{T}(\ldots x_1 + f_\theta^\mathcal{T}(x_0 + f_\theta^\mathcal{T}(x_0)\Delta t)\Delta t\ldots)\Delta t \tag{6}$$

where h is the evolution horizon. Therefore, the prediction error of $f_\theta^\mathcal{T}$ compounds and grows multiplicatively by every new prediction [33], [34]. This makes the dynamical system diverge away from the states present in the demonstration's trajectories, requiring the system to make predictions in states that are not supported by the training data, which is known as the *covariate shift* problem [32]. Consequently, the prediction error grows even larger.

An important reason for this issue to occur is that the learned system is expected to act over multiple steps when it is only being trained for predicting single steps. To alleviate this problem, the dynamical system must be trained for predicting multiple steps, by setting $H^i > 1$ and computing $x_{t,b}$ in (5) recursively, as shown in Fig. 4. In practice, however, the single-step loss is commonly employed, as the multistep loss has been regarded

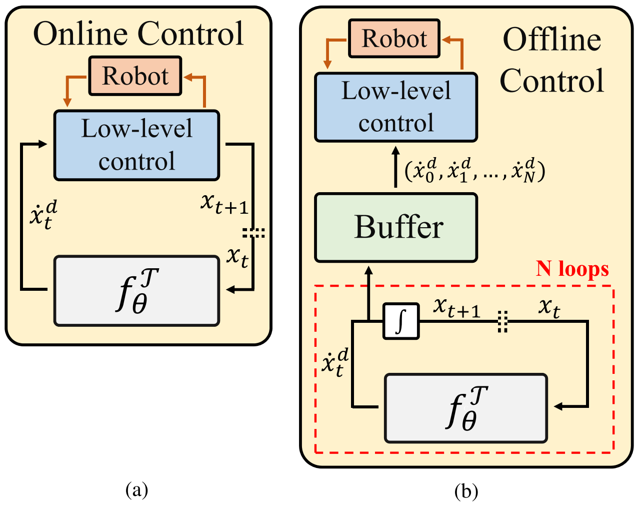

Fig. 5. Control strategies that can be used with CONDOR. (a) Online control. At every time step, $f_\theta^\mathcal{T}$ receives x_t from the sensors of the robot, and its output is fed to a low-level controller that tracks it. (b) Offline control. $f_\theta^\mathcal{T}$ is applied recursively N times to create, offline, a trajectory that is stored in a buffer. Afterward, the complete trajectory is tracked with a low-level controller.

as being challenging to optimize, even for short prediction horizons [35]. Nevertheless, these challenges can be addressed with current DNN optimization techniques.

Consequently, we optimize the multistep loss by noting that this can be achieved using *backpropagation through time* [36], [37], which has become popular and improved given its use in recurrent neural networks [38]. Hence, its limitations such as *exploding/vanishing gradients* or *ill-conditioning* [39] have been alleviated. Furthermore, specifically for this case, we can observe that every forward integration step in (6) can be interpreted as one group of layers inside a larger DNN that computes x_h. Then, each one of these groups has the same structure as the *residual blocks* in *ResNet* [40], which have also shown to be beneficial for alleviating vanishing/exploding gradients issues [41].

V. Simulated Experiments

In this section, we employ datasets of human handwriting motions to validate our method. Although these datasets contain human demonstrations, our evaluation of the learned motions is *simulated*, since no real system is involved in this process. This can be better understood with Fig. 5, where we show two different control strategies that can be employed with CONDOR. More specifically, Fig. 5(b) presents an offline control strategy where a trajectory is computed and stored in a buffer by applying CONDOR recursively. Afterwards, this trajectory is tracked by a low-level controller. In our evaluation, however, we ignore the low-level controller part and evaluate CONDOR using only the trajectory provided by the buffer, i.e., we assume that the trajectory is tracked perfectly. Despite this assumption, this methodology with this dataset has been extensively used in the literature, since it allows us to test if the learning method generates adequate state transition requests [2], [3], [7].

The DNN architecture and hyperparameter optimization process of the models used in this section are described in Appendices C and D, respectively.

A. LASA Dataset Validation: First-Order Two-Dimensional Motions

We validate our method using the LASA dataset,[4] which comprises 30 human handwriting motions. Each motion, captured with a tablet PC, includes 7 demonstrations of a desired trajectory from different initial positions. The state is represented as two-dimensional positions, and the learned systems are of first order, i.e., the output of $f_\theta^\mathcal{T}$ is a desired velocity. Although the demonstrations may have local intersections due to human inaccuracies, the shapes contained in this dataset can be well represented using first-order dynamical systems, which cannot represent intersections. Consequently, we employ the LASA dataset to evaluate motions modeled as first-order dynamical systems, which is the same approach that was taken by this article that introduced this dataset [2].

Fig. 6 shows three examples of dynamical systems learned with CONDOR. These motions share the following three features.

1) *Adequate generalization:* Motions generated in regions with no demonstrations smoothly generalize the behavior presented in the demonstrations.
2) *Accuracy*: The learned models accurately reproduce the demonstrations.
3) *Stability*: The vector fields suggest that, independently of the initial conditions, every motion reaches the goal.

In the following sections, we provide further details regarding each one of these points. Moreover, we compare CONDOR[5] with two other state-of-the-art methods for stable motion generation: 1) control lyapunov function-based dynamic movements (CLF-DM) using Gaussian mixture regression (GMR)[6] [8], and 2) stable dynamical system learning using Euclideanizing flows (SDS-EF)[7] [3]. CLF-DM (GMR) learns a dynamical system using a GMR and corrects its behavior whenever it is not stable according to a learned Lyapunov function. SDS-EF is a diffeomorphism shaping method, as introduced in Section IV-B1.

1) Generalization: Fig. 6 also depicts the performance of CLF-DM and SDS-EF on three motions. Here, we observe that even though the stability of these methods is guaranteed, unlike CONDOR, the behavior that they present in regions without demonstrations might not always be desired.

In the case of SDS-EF, unpredictable motions can be generated[8] (e.g., bottom image, bottom-right quadrant), which, furthermore, can reach very high speeds (e.g., 382 mm/s, while the demonstrations exhibit maximum speeds of around

[4][Online]. Available: https://cs.stanford.edu/people/khansari/download.html
[5]CONDOR code repository: https://github.com/rperezdattari/Stable-Motion-Primitives-via-Imitation-and-Contrastive-Learning
[6]CLF-DM code repository: https://github.com/rperezdattari/Learning-Stable-Motions-with-Lyapunov-Functions
[7]SDS-EF code repository: https://github.com/mrana6/euclideanizing_flows
[8]These results are not completely consistent with the ones reported in [3], since we removed additional preprocessing (smoothing and subsampling) to compare every method under the same conditions.

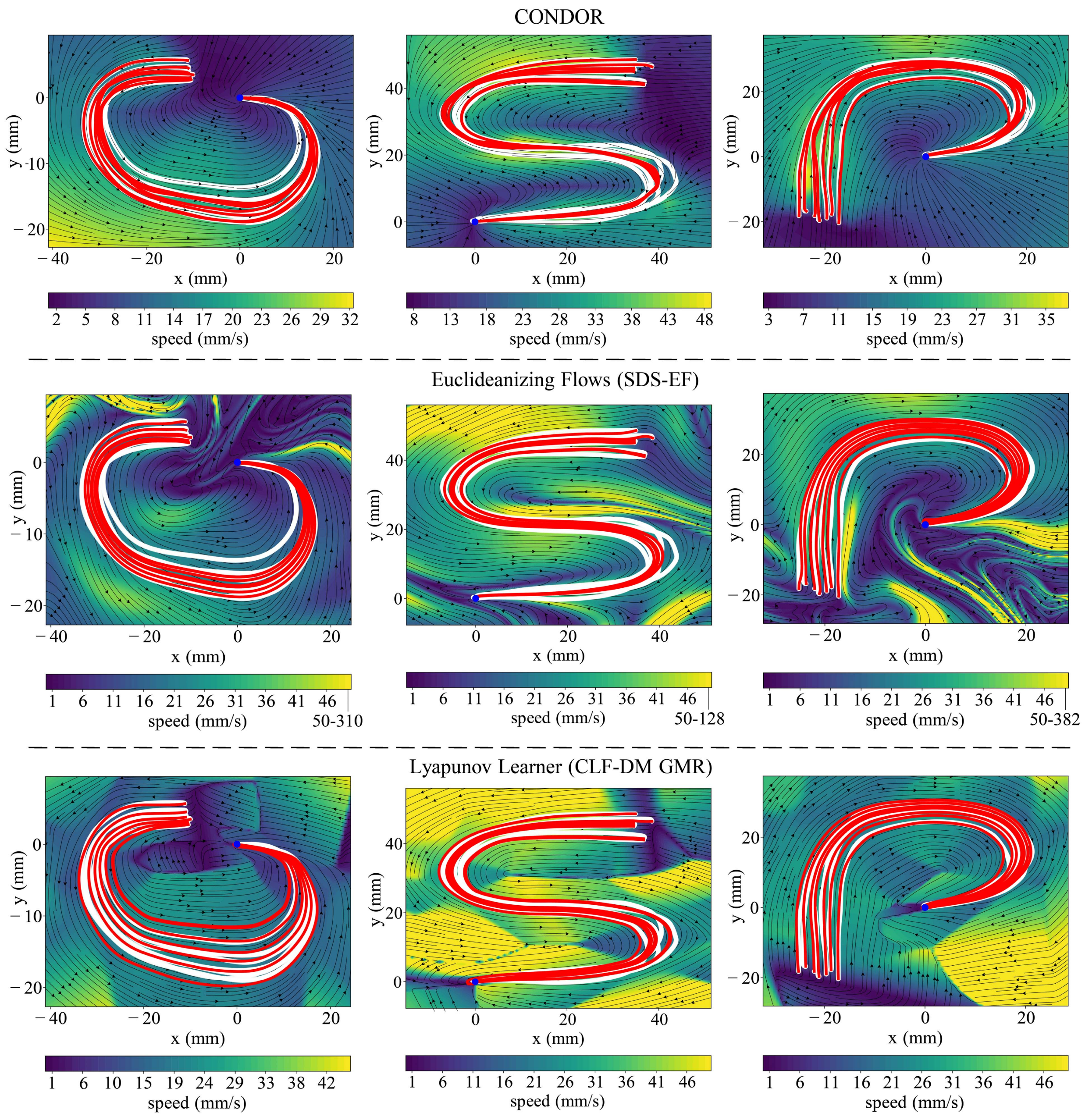

Fig. 6. Examples of LASA dataset motions learned using CONDOR, SDS-ES, and CLF-DS (GMR). White curves represent the demonstrations. Red curves represent the executed motions by the learned model when starting from the same initial positions as the demonstrations. The arrows indicate the vector field of the learned dynamical system (velocity outputs for every position). In SDS-ES, every speed greater than 50 mm/s is saturated to this value.

40 mm/s). Note, however, that this issue can be alleviated by optimizing SDS-EF for a shorter period of time, but this also makes it less accurate. Such unpredictability and high speeds can be a limitation in real-world scenarios. For instance, when humans interact with robots and must feel safe around them, or due to practical limitations, e.g., it is unfeasible to track the requested motions with a low-level controller.

Differently, in the case of CLF-DM, nonsmooth transitions are present in some regions of the state space due to the corrections applied by the Lyapunov function. This can also be a limitation, since robotic systems commonly avoid nonsmooth trajectories to minimize the risk of damage [19].

Lastly, Fig. 6 evidences that, in real-world scenarios, CLF-DM and SDS-EF are susceptible to making robots leave their workspaces. These methods do not constrain their trajectories to reside inside a specific space, they only guarantee that, eventually, these will converge to the goal. In practice, however, the learned trajectories might need to leave a robot's workspace to reach the goal. Then, in Fig. 6, if we assume that the observed regions are a robot's workspace, the vector fields of CLF-DM and SDS-EF indicate that some motions depart from it. In contrast, in CONDOR, the workspace is a positively invariant set w.r.t. the learned dynamical system (see Section IV-D); consequently, motions stay inside it.

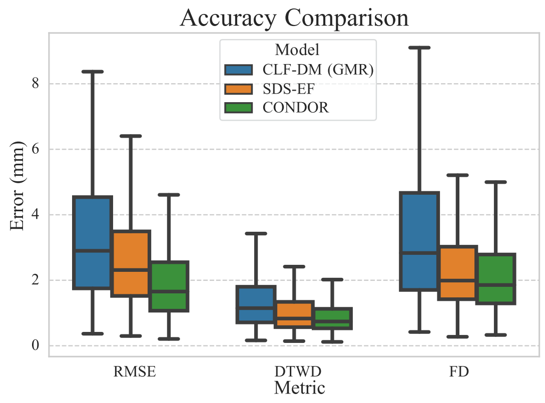

Fig. 7. Accuracy comparison of CONDOR against state-of-the-art methods. Each box plot summarizes performance over the 30 motions of the LASA dataset.

2) Accuracy: Fig. 6 indicates that every method is able to accurately reproduce the demonstrations. However, CLF-DM is clearly less accurate than CONDOR and SDS-EF. For instance, in the bottom-left image, the inner red trajectory drifts away from the demonstrations, coming back to them at the end of the motion due to its stability properties.

Quantitatively speaking, we can employ different metrics to evaluate the accuracy of the learned trajectories (see Fig. 7). Commonly, a distance between two trajectories is minimized; one trajectory corresponds to a demonstration, and the other corresponds to the one generated by the learned dynamical system when starting from the same initial condition as the demonstration. However, different distances between trajectories can be computed depending on the features that we aim to evaluate from the trajectories. To have a more complete view of the accuracy performance of our method, we compare CONDOR, CLF-DM (GMR),[9] and SDS-EF[10] under the following three distance metrics:

1) root mean squared error (RMSE);
2) dynamic time warping distance (DTWD) [42];
3) Frechet distance (FD) [43].

We can observe that CONDOR clearly achieves better results against CLF-DM (GMR) under every metric, while a smaller gap, yet superior, is achieved against SDS-EF.

3) Stability: Lastly, we quantitatively study the stability properties of CONDOR. As mentioned in Section IV, the stability of the motions it learns depends on the optimization problem being properly minimized. Therefore, we need to empirically test this after the optimization process finishes.

To achieve this, we integrate the dynamical system for L time steps, starting from P initial states, and check if the system converges to the goal (i.e., fixed-point iteration, where the fixed point corresponds to the goal). The larger the P, the more accurate the results we obtain. If L is large enough, the system should converge to the goal after L steps. Hence, by computing the distance between the last visited state and the goal, and checking that it is below some predefined threshold ϵ, it is possible to evaluate if a trajectory is successful or not (i.e., if it converges to the goal).

We evaluated CONDOR using all of the motions present in the LASA dataset with $L = 2000$ and $P = 1225$ with $\epsilon = 1$ mm, and observed that 100% of the trajectories reached the goal. Hence, CONDOR is able to successfully learn stable motions.

[9]Each GMR consisted of 10 Gaussians and each Lyapunov function was estimated using three asymmetric quadratic functions.

[10]Results were extracted from [3].

B. Ablation Study

To better understand CONDOR and the relevance of its different parts, we perform an ablation study where we compare the following four variations of this method.

1) *CONDOR*: The base method studied in Section V-A.
2) *CONDOR (relaxed)*: The contrastive loss for stability is replaced with the triplet loss. This variation is presented to observe the importance of minimizing the exact loss presented in (1) or whether it is enough to enforce this type of structure in the latent space of the NN to obtain stable motions.
3) *CONDOR (fixed gains)*: As explained in Section IV-E1, having adaptive gains in the latent dynamical system described in (2) should help obtaining more flexible motions. Therefore, this model, with fixed gains, is studied to observe the relevance and effect of using adaptive gains.
4) *BC:* The stability loss is removed and only the BC loss is employed to learn motions. This model is used to study the effect that the stability loss can have on the accuracy of the learned motions. To observe the behavior of BC, we refer the reader to Fig. 2.

Fig. 8 showcases examples of motions learned with both CONDOR (relaxed) and CONDOR (fixed gains). In both cases, the motions display accuracy and stability. However, these models differ in their generalization. CONDOR (relaxed) has a generalization behavior similar to the one of CONDOR shown in Fig. 6. In these cases, the regions of the state space without demonstrations exhibit a trend that resembles the one observed in the demonstrations. In contrast, the generalization of CONDOR (fixed gains) does not follow this trend as closely. For example, within certain regions, the velocity of the motions decreases, and as they approach the demonstrations, their direction becomes nearly orthogonal, indicating a discrepancy between the generalized behavior and the pattern presented in the demonstrations.

1) Accuracy: In this section, we compare the accuracy of the different variations of CONDOR (see Fig. 9). Intuitively, BC should perform better than any variation of CONDOR, since it only optimizes the BC loss. In contrast, the other variations also optimize the stability loss, which could harm/limit the minimization of the BC loss. Consequently, BC is the lower bound for the accuracy performance, i.e., best case scenario, a variation of CONDOR performs as well as BC does.

In Fig. 9, we observe that the accuracy performance of all of the variations of CONDOR is very similar, including the BC case. This result shows that CONDOR, and its variations, is able to effectively minimize the BC and stability loss together without harming the accuracy performance of the learned motions.

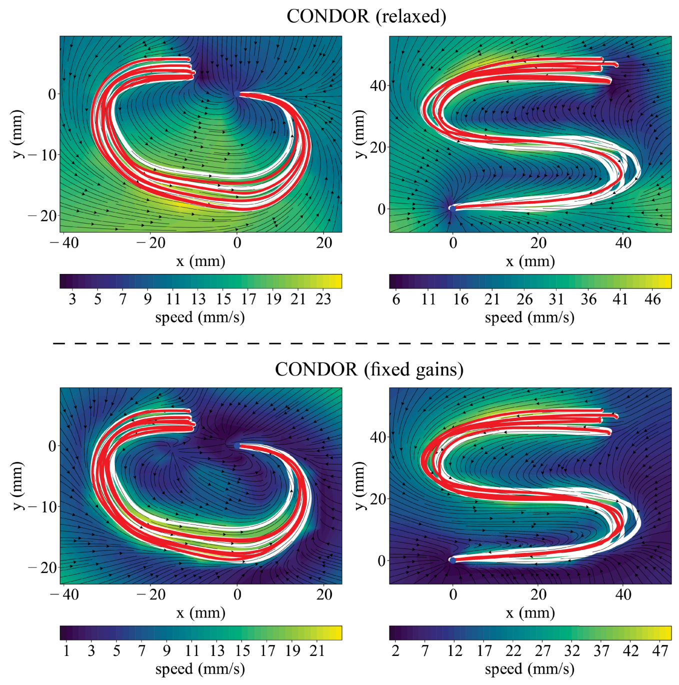

Fig. 8. Examples of LASA dataset motions learned using two variations of CONDOR: 1) relaxed and 2) fixed gains.

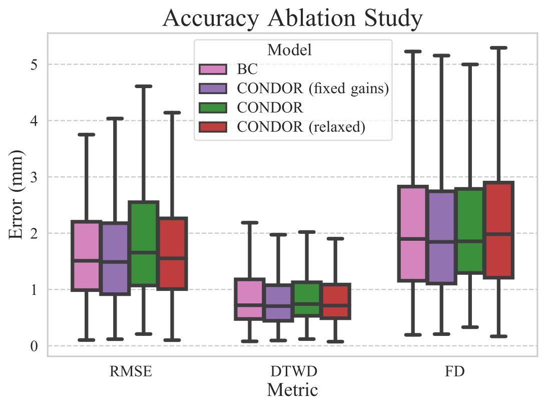

Fig. 9. Accuracy comparison of different variations of CONDOR. Each box plot summarizes performance over the 30 motions of the LASA dataset.

TABLE I
PERCENTAGE OF UNSUCCESSFUL TRAJECTORIES OVER THE LASA DATASET ($L = 2000$, $P = 1225$, $\epsilon = 1$ mm)

BC	CONDOR (fixed gains)	CONDOR (relaxed)	CONDOR
36.4653%	0.0054%	**0.0000%**	**0.0000%**

The bold entities indicate the best results.

2) Stability: Table I shows the results of stability tests on the presented methods. We can observe that when stability is not enforced (i.e., BC), the percentage of unsuccessful trajectories is significant, being larger than one-third of the total amount of trajectories. In contrast, when stability is enforced using adaptive gains, the system achieves perfect performance (i.e., every trajectory reaches the goal), as it is observed with the results of CONDOR and CONDOR (relaxed). Interestingly, this result also shows that the relaxed variation of CONDOR can be employed for achieving stable motions without having a loss in performance. In contrast, when fixed gains are employed, although the percentage of unsuccessful trajectories is very low ($< 0.01\%$), the performance degrades. This result suggests that the stability loss is not being as effectively minimized as when the gains are adaptive.

3) Dynamical Systems Mismatch: So far, we observed that every variation of CONDOR that minimizes the stabilization loss is able to learn accurate and stable motions. The case of CONDOR (fixed gains) showed a slightly worse stability performance and poorer generalization capabilities than CONDOR and CONDOR (relaxed). However, CONDOR has not shown to be clearly superior to its variations, especially in the CONDOR (relaxed) case.

Since CONDOR (relaxed) approximates the loss that minimizes the distance between $y_t^{\mathcal{L}}$ and $y_t^{\mathcal{T}}$, the trajectories that it obtains with $f^{\mathcal{L}}$ and $f_\theta^{\mathcal{T}}$ in the latent space should diverge faster than the ones generated with CONDOR. To investigate this idea, we evaluate the optimization of this loss by separately simulating $f_\theta^{\mathcal{T}}$ and $f^{\mathcal{L}}$ when starting from the same initial conditions. If the stabilization loss is perfectly minimized, these simulations should yield the same trajectories when mapping the evolution of $f^{\mathcal{L}}$ to task space;[11] otherwise, they should diverge from each other.

Fig. 10 presents motions learned using the different variations of CONDOR and shows motions generated in task space when following $f_\theta^{\mathcal{T}}$ and $f^{\mathcal{L}}$. We can observe that CONDOR performs well in the complete state space, where, for most trajectories, it is not possible to detect a difference between the results obtained using $f^{\mathcal{L}}$ and $f_\theta^{\mathcal{T}}$. In contrast, we can observe that, for the other cases, trajectories diverge more pronouncedly. Interestingly, the divergent trajectories seem to overlap with the regions where demonstrations are provided. This suggests the stabilization loss is not properly minimized in this region, indicating that these variations of CONDOR struggle to find good solutions in the regions of the state space where the imitation and stabilization losses are optimized together, i.e., in the demonstrations. Finally, it is also possible to observe that CONDOR (relaxed) obtains trajectories that are slightly more similar to the ones obtained with CONDOR (fixed gains), although it is not conclusive.

Quantitatively, we can analyze this trajectory difference by computing the accumulated error between the trajectories generated using both dynamical systems. Fig. 11 shows this error as a function of the trajectory length. As expected, this error grows for the dynamical systems as a function of their length. However, CONDOR obtains a significantly lower error than its variations. Furthermore, Fig. 11 clearly shows that CONDOR (relaxed) outperforms CONDOR (fixed gains). Finally, since this accumulated error is a consequence of how well the stability loss is minimized, these results might explain why the stability performance of CONDOR (fixed gains) is not perfect, i.e., this

[11]Trajectories from $f^{\mathcal{L}}$ with known initial conditions in $\mathcal{T}$ (hence, $y_0^{\mathcal{L}} = \psi_\theta(x_0)$), can be mapped to $\mathcal{T}$ by recursively applying $x_{t+1} = x_t + \phi_\theta(y_t^{\mathcal{L}})\Delta t$.

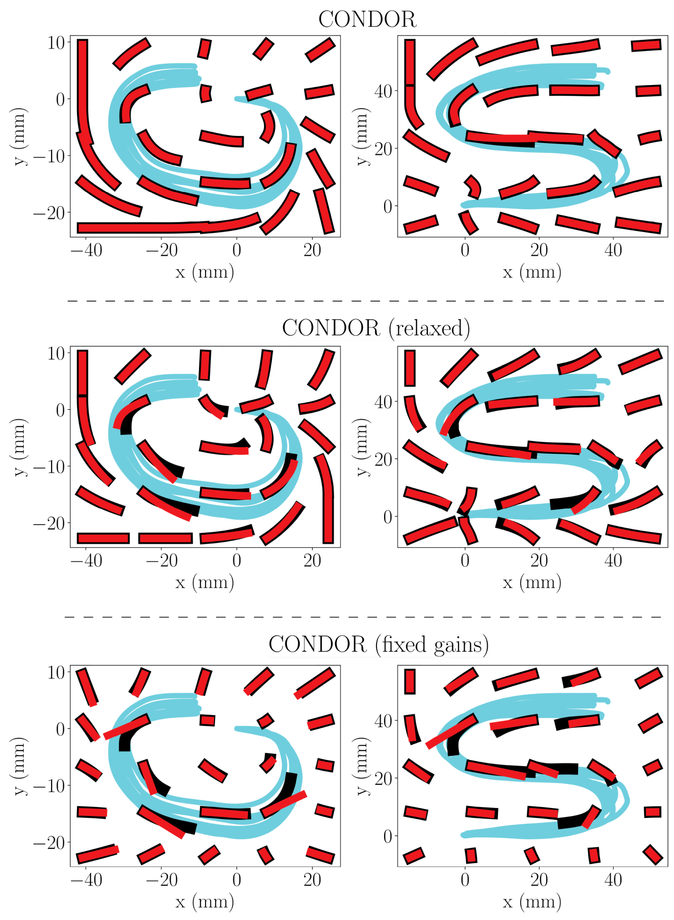

Fig. 10. Dynamical systems mismatch comparison. Blue curves represent the demonstrations, black curves represent trajectories generated using $f_\theta^{\mathcal{T}}$ of the dynamical system, and red curves represent trajectories generated using $f^{\mathcal{L}}$ of the dynamical system. The black and red trajectories were obtained by integrating the dynamical system through 80 time steps.

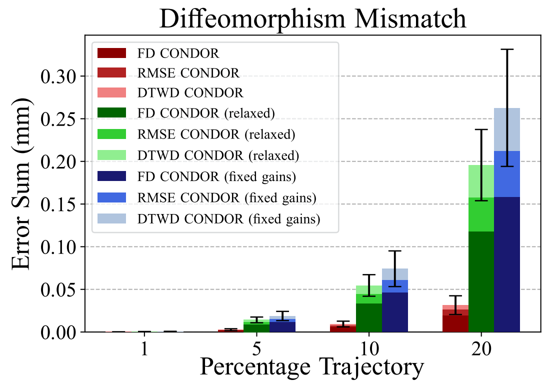

Fig. 11. Trajectory mismatch error. Results are presented as a function of the length of a trajectory with respect to the complete length of the demonstrated trajectories, corresponding to 1000 transitions. 100 trajectories in the task and latent spaces were simulated, whose initial positions were uniformly distributed in the motion's state space. The results show the mean and half of the standard deviation of the error computed with these simulations.

variation is not able to successfully minimize the stability loss in the complete state space.

As a final remark, we can note that generating trajectories in the latent space and then mapping them to task space can have other applications, such as predicting future states efficiently and employing them, for instance, in model predictive control frameworks. It is considerably faster to generate trajectories in the latent space than using the complete DNN architecture to compute them in task space, since the number of parameters and layers required to do so is smaller. Then, once the trajectory is generated in the latent space, it can be mapped to task space as one batch in one forward pass.

C. *LAIR Dataset Validation: Second-Order 2-D Motions*

In this section, we introduce the *LAIR handwriting dataset*. The objective is to test the accuracy and stability performance of the proposed method for second-order motions, where the state comprises both position and velocity. This dataset contains ten human handwriting motions collected using a mouse interface on a PC. The state here is four-dimensional, encompassing a two-dimensional position and velocity, and the output of $f_\theta^{\mathcal{T}}$ is the desired acceleration. The dataset's shapes present several position intersections that have been designed to require, at least, second-order systems to model them. This dataset is employed to test the scalability of the proposed method in terms of the order of the motion.

Unlike the LASA dataset, the LAIR dataset contains raw demonstrations without any type of postprocessing. Hence, the ending points of the demonstrated trajectories might not always coincide exactly. To account for this, the goal of a motion is computed by taking the mean between these ending points.

1) Accuracy: Fig. 12 shows three examples of motions of the LAIR dataset. These motions can only be modeled using a dynamical system of, at least, second order. First-order systems only employ position information to generate a trajectory; hence, visiting the same position two times will generate an ambiguity for the learning algorithm. This makes the learned system collapse to a solution that lies in between the multiple demonstrated options. Therefore, we observe that first-order systems with CONDOR are not able to appropriately model the shown motions.

In contrast, we observe that second-order systems are able to appropriately capture the dynamics of the demonstrated motions and execute them as they were intended. However, some trajectories (especially those coming from the tail of the initial-state sampling distribution) do not go through the first intersection, since they start from a position that, given its distance from the initial states of the demonstrations, directly follows the trend of the motion after this intersection. If this is a limitation for a specific application, providing demonstrations in those regions would make the system behave as expected. Finally, another interesting feature of these motions, is that the different trajectories, eventually, seem to collapse to the same position, overlapping with each other. This comes as an artifact when

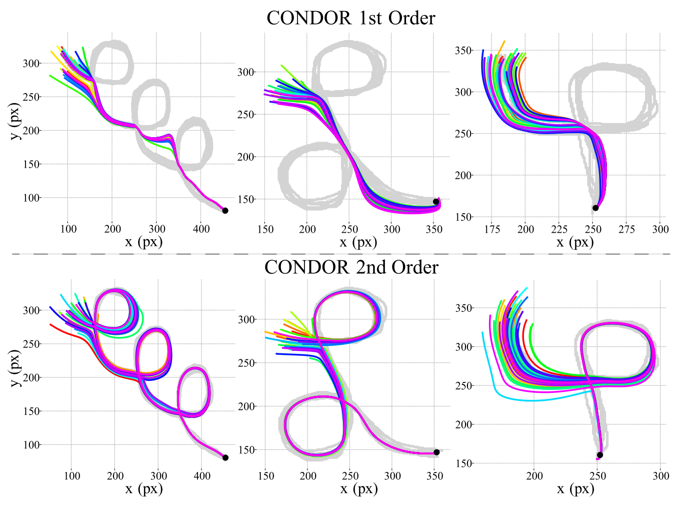

Fig. 12. Motions modeled using CONDOR with first-order and second-order systems. The shapes in grey correspond to the demonstrations. The colored curves correspond to different instances of trajectories generated when starting from different initial states. Every trajectory was initialized with zero velocity and the initial positions were obtained by sampling from a Gaussian distribution around the initial positions observed in the demonstrations. 36 trajectories were sampled per plot.

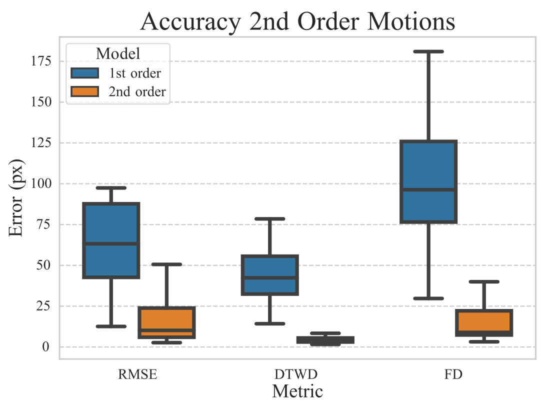

Fig. 13. Accuracy comparison of CONDOR when modeling motions using first-order and second-order systems. Each box plot summarizes performance over the 10 motions of the LAIR dataset.

incorporating the stability loss, where the systems find these solutions to ensure stability.

Quantitatively, the same conclusions can be drawn when observing Fig. 13. This figure presents the results of the accuracy of both CONDOR variations under the same metrics employed in Section V-A2. As expected, the second-order systems outperform the first-order systems by a large margin.

2) Stability: Finally, we study the stability of the motions generated with CONDOR over the LAIR dataset when using first-order and second-order systems. Table II shows that when using first-order systems, CONDOR struggles to generate stable motions with second-order demonstrations. For instance, when a demonstration has a *loop*, the optimization of the DNN might not find a proper solution, since trajectories inside the loop do not have a way of reaching a region outside the loop without ignoring

TABLE II
PERCENTAGE OF SUCCESSFUL TRAJECTORIES OVER THE LAIR DATASET ($L = 2000$, $P = 1225$, $\epsilon = 10$ px)

CONDOR (first order)	CONDOR (second order)
9.3388%	**0.0000%**

The bold entities indicate the best results.

TABLE III
CHARACTERISTICS OF THE REAL-WORLD EXPERIMENTS

Task	Dim state	Order motion	Control	Data collection	# demos
Hammer hanging	3	1	online, end eff.	motion capture	10
Writing two	4	2	offline, end eff.	computer mouse interface	6
Table cleaning	6	1	online, joints	kinesthetic teaching	2*

* Two motion models were learned, and one demonstration was used per model.

the demonstrations. In contrast, CONDOR with second-order systems is always able to learn stable motions.

VI. Real-World Experiments

To validate the proposed framework in more realistic scenarios, we design the following three real-world experiments using a 7-DoF KUKA iiwa manipulator:

1) hammer hanging;
2) writing the number two;
3) cleaning a table (see Fig. 14).

Throughout these experiments, the following four important characteristics of the learning problem are changed:

1) dimensionality of the motion;
2) order of the motion;
3) control strategy;
4) data collection method.

These characteristics define different IL scenarios that can be found in real-world robotic problems. Hence, by testing CONDOR in these scenarios, we aim to show the applicability, flexibility, and robustness of our method. Furthermore, if we compare these scenarios with the simulated ones studied in the previous sections, we can observe that our method is not restricted to two-dimensional motions only and that it can also work in higher dimensional problems. Table III shows a summary of the real-world experiments, which are explained in detail in the following sections.

Similarly to the LAIR dataset, the demonstrations are not postprocessed in these experiments. Hence, in this section, the goal of the motions is also computed by taking the mean between the ending points of each demonstration.

A. Hammer Hanging: First-Order 3-D Motions

This experiment consists of learning to control the end-effector's position of a robot such that it hangs a hammer [see Fig. 14(a)], allowing us to test the behavior of CONDOR for

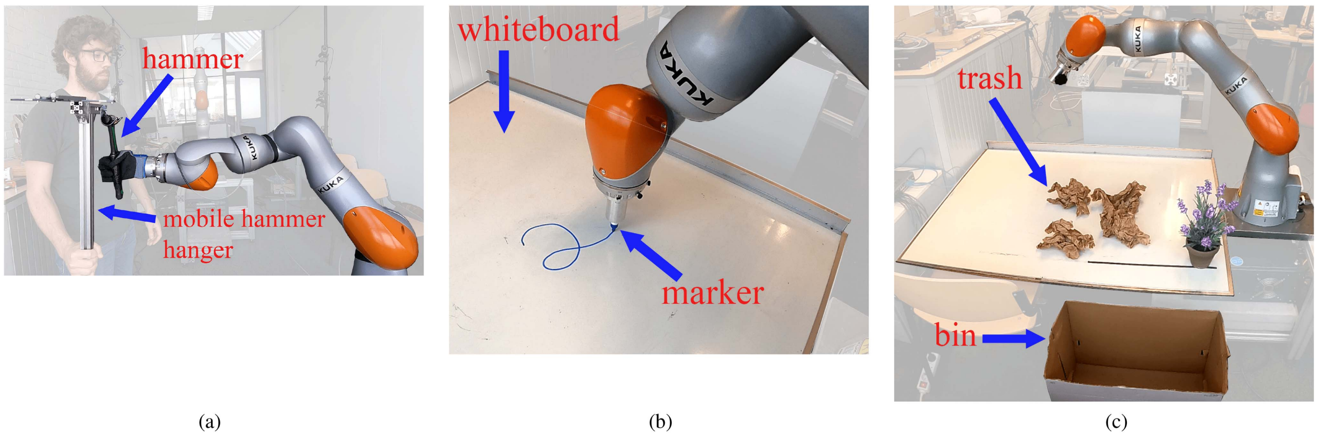

Fig. 14. Setup of real-world experiments. (a) Hammer hanging. (b) Writing number two. (c) Table cleaning.

first-order 3-D motions. This problem is interesting since it shows that implicit knowledge that otherwise requires modeling, can be transferred to the robot via human demonstrations. In this case, this knowledge includes information about the geometry of the hammer and the hanger that is required to hang the hammer.

1) Control: We employ the online control strategy depicted in Fig. 5(a), which allows the robot to be reactive to perturbations and adjust its motion *on the fly* if the environment changes. Hence, at every time step, the robot obtains its position with respect to the goal (i.e., the hanger) and sends an end effector's velocity request to a low-level controller. For details regarding the low-level controller, the reader is referred to Appendix E-B.

2) Demonstrations: We used a motion capture system to collect demonstrations. The demonstrator had to wear a glove whose position was tracked by the tracking system. We recorded 10 demonstrations and used them to train CONDOR.

This approach has the advantage of it being comfortable for the human, since it does not require the human to adjust to any specific interface nor interact with the robot, which can require training. Nevertheless, since the robot embodiment is not being employed to collect the demonstrations, there is no guarantee that the collected motions will feasible for the robot to execute. Therefore, it is necessary to record motions that can be executed by the robot, which, depending on the problem, might require knowledge about the robotic platform.

3) Moving Goal: To test the reactive capabilities of this approach, and the generalization properties of motions modeled as dynamical systems, we made this problem more challenging by making the hanger movable. To achieve this, we added tracking markers to the top of the hanger and fed the hanger's position to CONDOR in real time. Consequently, while the robot was executing the hanging motion, the hanger could be displaced and the robot had to react to these changes in the environment.

Notably, no extra data is required to achieve this, since the motion of CONDOR is computed as a function of the relative position of the robot w.r.t. the goal. Hence, by displacing the goal, the position of the end effector with respect to the hanger changes, making CONDOR provide a velocity request according to this new position.

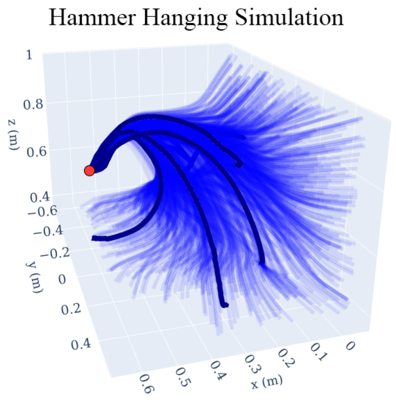

Fig. 15. Blue trajectories represent the learned model's evolution of the robot end effector's position when starting from different initial conditions. The larger and darker trajectories correspond to demonstrations. Some demonstrations are occluded and others were removed for visualization purposes. The red point corresponds to the goal.

4) Results: Fig. 15 shows a 3-D plot with 1250 simulated trajectories generated with CONDOR when starting from different initial positions. We can observe that all of the trajectories reach the goal while following the shape in the demonstrations. The performance of this model on the real robot can be observed in the attached video.

B. Writing: Second-Order 2-D Motions

We also tested CONDOR in a writing scenario [see Fig. 14(b)]. The objective is to control the robot's end effector to write the number two on a whiteboard. To write the number two, it is necessary to use second-order motions, since this character has one intersection. Therefore, in this experiment, we aim to validate the ability of CONDOR for modeling second-order motions. Finally, note that for writing it is only necessary for the robot to move in a two-dimensional plane; however, since the motion is of second order, the state space of the robot is four-dimensional (the same as the motions in the LAIR dataset).

1) Control: We employ the offline control strategy, as depicted in Fig. 5(b). This approach is suitable for writing since in this task it is important that the trajectory that the robot executes is consistent with the one that CONDOR predicts from the initial state. For instance, if the robot, while executing the motion is perturbed by its interaction with the whiteboard in some direction, it would transition to a state that is not consistent anymore with the character that has been written so far. In an offline control approach this is not very critical, because, given that the reference of the motion is precomputed, it would make the robot move back to a state that is consistent with the motion that is being written. For details regarding the low-level controller, the reader is referred to Appendix E-C.

2) Demonstrations: The same PC mouse interface developed to collect the LAIR dataset is employed here. Six demonstrations were collected and used to train CONDOR.

3) Results: The simulated results of this experiment follow the same behavior as the ones presented in Section V-C. To observe its behavior on the real robot, the reader is referred to the attached video.

C. Table Cleaning: First-Order 6-D Motions

Finally, we test CONDOR in a cleaning task. The objective is to use the robot's arm to push garbage, which is on top of a table, to a trash bin. Differently from the other scenarios, in this case, the robot's joint space is directly controlled with CONDOR. Hence, we learn a 6-D motion. Note that the robot has 7 degrees of freedom, but we keep the last joint fixed as it has no influence on the task.

Since the motions learned by CONDOR can be used as primitives of a more complex motion, in this experiment, we highlight this capability by learning two motions that are sequenced together to generate the complete cleaning behavior. Each motion is trained with only one demonstration.

This scenario allows us to test two features of our method: 1) its behavior in a higher dimensional space (6-D), and 2) its capability to learn motions from only one demonstration.

1) Control: Similarly to the hanging hammer experiment, we use the online control strategy [see Fig. 5(a)]. Differently than before, in this case, the joint space of the robot is directly controlled with CONDOR, i.e., a reference velocity for the joints is provided to the low-level controller. Joint-space control is suitable for this task because the configuration of the robot is important for completing the task successfully since its body is used to push the trash. For details regarding the low-level controller, the reader is referred to Appendix E1.

2) Demonstrations: For this experiment, kinesthetic teaching was used to collect demonstrations. This approach consists of collecting demonstrations by physically interacting with the robot and guiding it along the desired trajectory. To make this task easier, the gravitational forces of the robot were compensated such that it would not move unless the human interacted with it.

3) Results: Fig. 16 presents simulated results of the second cleaning primitive learned in this experiment. Since the motion is six-dimensional and, hence, it is very challenging to visualize in one plot, we use six different plots to separately show the evolution of each state dimension as a function of time.

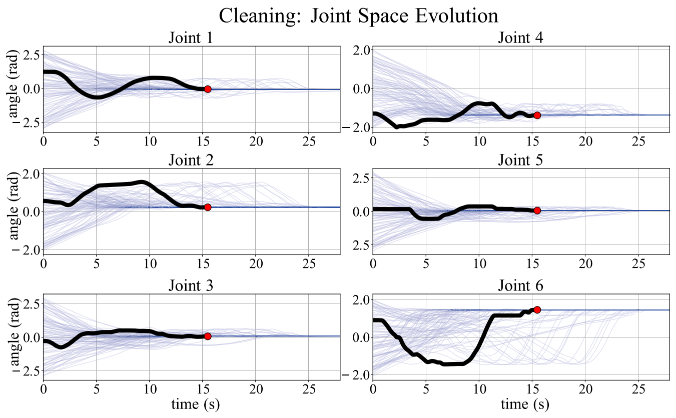

Fig. 16. Simulated trajectories, as a function of time, of the second motion of the cleaning task. Blue trajectories correspond to evaluations of the model under different initial conditions and the black trajectory corresponds to the demonstration. The red point is the goal.

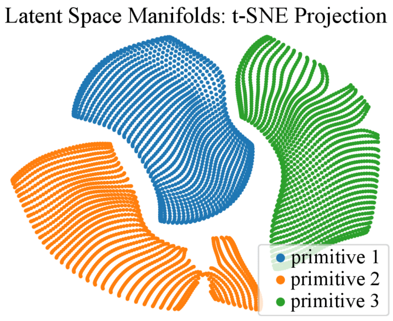

Fig. 17. t-SNE projection of the motion manifolds present in the latent space of the DNN.

In this case, we simulate 100 trajectories using CONDOR, since more make the plots difficult to analyze. From them, we observe that as time increases, every trajectory eventually reaches the goal. Note that, given that their initial states are random, they can start further away or closer to the goal than the demonstration; therefore, it might take them a longer/shorter time to reach the goal. Lastly, we can observe that the demonstrated trajectory and some simulations, either overlap or have the same shape with a phase shift, which showcases that the demonstrated behavior is captured by CONDOR.

The reader is referred to the attached video to observe the behavior of the cleaning primitives on the real robot.

VII. Extending CONDOR

One of the advantages of our proposed framework is that we can extend it to address more complex problems. Therefore, there are interesting areas of research that can be studied with CONDOR. In this section, we aim to show the steps that we have taken in this direction, which we plan to study deeper in future work. More specifically, we tested the following two extensions.

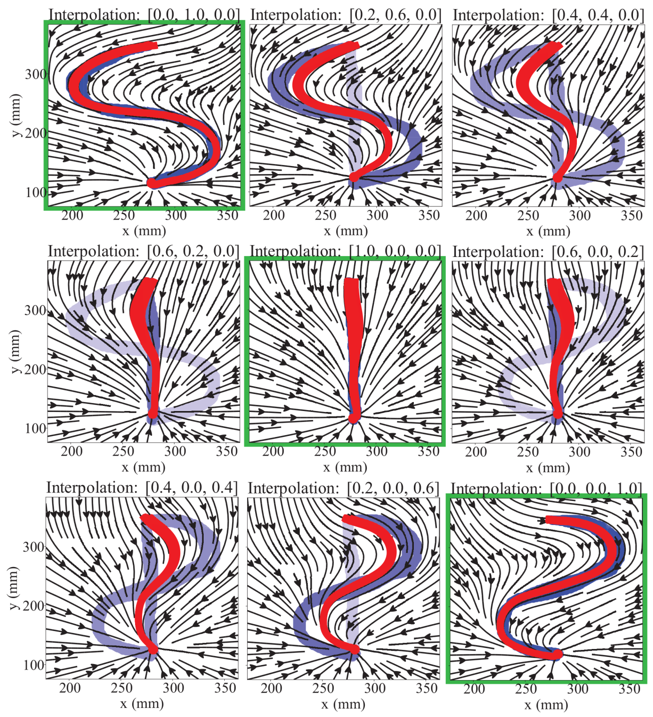

Fig. 18. Different motions learned in one DNN. Blue curves correspond to demonstrations, the darker their color, the more they influence each motion. Arrows represent the vector field of the motion, and red curves correspond to simulations of trajectories executed by the model. The figures highlighted in green correspond to the cases where the BC loss is minimized. The remaining figures correspond to interpolated motions. In the title of each plot, we can observe the code provided to the network to generate each motion.

1) *Obstacle avoidance:* Multiple obstacle avoidance methods have been proposed for motions modeled as dynamical systems, and it is an active field of research [44], [45], [46], [47]. We test CONDOR with one of these extensions [44] and observe that it works properly. However, apart from this validation, we do not provide a contribution to this problem. Hence, the reader is referred to Appendix F for more details regarding obstacle avoidance in this research.
2) *Multimotion learning and interpolating:* Another interesting field of research is learning multiple motions together in one neural network model. This allows interpolating between these motions, generating novel behaviors that are not present in the demonstrations. This can, for instance, reduce the number of human demonstrations required to learn and generalize a problem to a different situation.

In the following section, we study the interpolation capabilities of motions learned with CONDOR.

A. Multimotion Learning and Interpolation

We aim to provide preliminary results regarding the multimotion learning capabilities of CONDOR, and its behavior in terms of interpolation and stability. To learn multiple motions in one neural network, we extend its input with a one-hot code that indicates, which motion is selected. For instance, if we learn three motions, we have three codes $[1, 0, 0]$, $[0, 1, 0]$, and $[0, 0, 1]$. Then, each code is used together with a different set of demonstrations to optimize ℓ_{bc}. To interpolate between these motions, we select an input code of the DNN that has an intermediate value between the ones of the motions, e.g., $[0.5, 0.5, 0.0]$. To ensure stability for all of the interpolated motions, we can minimize ℓ_{stable} for each motion and also for the ones in the interpolation space, which should create a bijective mapping between the complete input of the DNN (state and code) and its latent space. Part of this can be observed in Fig. 17, which shows a t-SNE [48] projection of three manifolds corresponding to the mapping of the state space of three motions to the DNN's latent space. Since each motion is mapped to a different region of the latent space, it is possible to move in between these regions to create interpolated motions.

Fig. 18 shows these motions and some examples of their interpolation. We can note that the interpolation works properly, where features of different motions are combined to create novel behaviors. Furthermore, we observe that, as expected, the closer to a motion we interpolate, the more features of this motion the interpolated one showcase. Finally, regarding stability, every motion has zero unsuccessful trajectories with $L = 2000$, $P = 1225$, and $\epsilon = 10$ px.

VIII. Conclusion

In the context of robotic reaching motions modeled as dynamical systems, we introduce a novel contrastive loss that extends current IL frameworks to achieve globally asymptotically stable behaviors. We optimize this loss together with a BC loss, which, despite its practical limitations due to the covariate shift problem, can achieve state-of-the-art results by minimizing the multistep loss instead of the single-step loss, as observed in our experiments. Importantly, our stability loss can also be employed with other IL approaches, though its effectiveness with other losses remains untested.

Further experiments demonstrate that our framework, CONDOR, can effectively learn stable and accurate motions across various scenarios. These experiments were conducted in both simulated settings and with a real robot. We observed that CONDOR learns successful behaviors in the following:

1) two-dimensional first-order motions (LASA dataset);
2) three-dimensional first-order motions (hammer hanging);
3) four-dimensional second-order motions (LAIR dataset and writing two);
4) six-dimensional motions (cleaning table).

Lastly, we observe that CONDOR can be extended to learn multiple motions and interpolate between them, allowing it to generate more stable behaviors without requiring more demonstrations. This interesting area of research will be explored further in future work.

While this paper's findings are promising, they also reveal limitations that inspire other future research directions. First, our method has only been tested on relatively low-dimensional state spaces; its applicability in higher dimensional spaces remains unexplored. Additionally, we assume that the employed state representations used are minimal, a condition not always met in robotics. For instance, orientation representations often employ non-Euclidean manifolds (e.g., unit quaternions or rotation matrices) that introduce constraints, making the state representations nonminimal [49]. A further assumption is the existence of a low-level controller capable of generating the state transitions requested by CONDOR. While this is reasonable for manipulators, it can be a limiting factor in highly underactuated robots. Finally, CONDOR assumes that a proper state estimation (e.g., robot pose, goal location, or obstacles) is achieved by other modules.

Appendix A
Approximating a Diffeomorphism

For achieving stable motions, CONDOR optimizes ℓ_{stable}. This loss is designed to enforce the conditions of Theorem 1 in the NN employed to represent the dynamical system $f_\theta^{\mathcal{T}}$. In this section, we show that as a consequence of this, ψ_θ approximates a diffeomorphism when $\mathcal{T}$ is a Euclidean space.

Definition 1 (Diffeomorphism): A mapping between two manifolds $\psi_\theta : \mathcal{T} \rightarrow \mathcal{L}$ is called a diffeomorphism if it is differentiable and bijective.

In general, we can consider neural networks to be differentiable, since most of the employed activation functions are differentiable. However, there are some exceptions to this, as it is for the case of the ReLU activation [50]. In practice, these exceptions are nondifferentiable at a small number of points and they have right and left derivatives, so they do not present many issues when computing their gradients. However, strictly speaking, for such cases, our method would make ψ_θ converge to a *homeomorphism* instead. Differently to a diffeomorphism, a homeomorphism only requires the mapping to be continuous, but not differentiable. Nevertheless, for simplicity, we will assume that ψ_θ is differentiable.

Then, we need to study if ψ_θ converges to a bijective function to conclude that it approximates a diffeomorphism.

Definition 2 (Bijective function): A function is bijective if it is injective and surjective.

Definition 3 (Injective function): A function is injective if every distinct element of its domain maps to a distinct element, i.e., ψ_θ is injective if $\psi_\theta(x_a) = \psi_\theta(x_b) \Rightarrow x_a = x_b, \forall x \in \mathcal{T}$.

Definition 4 (Surjective function): A function is surjective if every element of the function's codomain is the image of at least one element of its domain, i.e., $\forall y \in \mathcal{L}, \exists x \in \mathcal{T}$ such that $y = \psi_\theta(x)$.

From these definitions, it is clear that the surjectivity of ψ_θ is straightforward to show, since it depends on how its codomain is defined. In this work, we define the codomain of ψ_θ as $\mathcal{L}$, which is the manifold resulting from the image of ψ_θ. In other words, the codomain $\mathcal{L}$ is a set that only contains the outputs of ψ_θ produced from $\mathcal{T}$. In such cases, a function is surjective, since its codomain and image are equal, which ensures that $\forall y \in \mathcal{L}, \exists x \in \mathcal{T}$ such that $y = \psi_\theta(x)$ (Definition 4).

Consequently, it only remains to prove that if the conditions of Theorem 1 are met, then ψ_θ is injective when $\mathcal{T}$ is a Euclidean space.

Proposition 2: If the conditions of Theorem 1 are met and the domain of ψ_θ is $\mathcal{T}$; then, ψ_θ is injective.

Proof: By contradiction, let us assume that these conditions are met and ψ_θ is not injective. Then, let us take two elements of $\mathcal{T}$, x_{a_0} and x_{b_0}, where $x_{a_0} \neq x_{b_0}$. If ψ_θ is not injective, there $\exists x_{a_0}$ and $\exists x_{b_0}$, such that $\psi_\theta(x_{a_0}) = \psi_\theta(x_{b_0})$. In such cases, from Condition 1) of Theorem 1, we know that as $t \rightarrow \infty$, the mappings of these elements will generate the same trajectory in $\mathcal{L}$ following the dynamical system $f^{\mathcal{L}}$, which converges to y_g.

Since the evolution of the variables x_{a_0} and x_{b_0} is completely defined by the evolution of their mappings in $\mathcal{L}$ (i.e., $\dot{x} = \phi(y)$), both variables will present the same time derivative in $\mathcal{T}$. Consequently, given that the trajectories obtained when starting from x_{a_0} and x_{b_0}, at time t, are described by $x_i(t) = x_{i_0} + \int_{\tau=0}^{t} f(x(\tau))d\tau$,[12] where $i \in \{a, b\}$, their integral part will be the same $\forall t$. Then, $\forall t$ the distance between both trajectories is $d(x_{a_0}, x_{b_0}) = ||x_a(t) - x_b(t)|| = ||x_{a_0} - x_{b_0}||$, where $d(\cdot, \cdot)$ is a distance function.

Thus, as $t \rightarrow \infty$, x_a and x_b will converge to two different points x_{a_g} and x_{b_g}, respectively. However, we also stated that their respective mappings converge to y_g, i.e., $\psi_\theta(x_{a_g}) = \psi_\theta(x_{b_g}) = y_g$. In this case, Condition 2) of Theorem 1 implies that $x_{a_g} = x_{b_g} = x_g$. This contradicts the fact that $x_g^a \neq x_g^b$. Consequently, ψ_θ is injective. □

Finally, from Proposition 2, we can conclude that if the conditions of Theorem 1 are enforced in $f_\theta^{\mathcal{T}}$; then, ψ_θ will approximate a diffeomorphism.

Appendix B
Stability of $f^{\mathcal{L}}$ With Adaptive Gains

In this section, we show that y_g is globally asymptotically stable in the system introduced in (2) when the adaptive gain $\alpha(y_t)$ is greater than zero. Note that the derivation introduced here is analogous to the one of the discrete-time case when the system is simulated using the forward Euler integration method. However, in the latter, the condition for global asymptotic stability is $0 < \alpha(y_t) < 2/\Delta t$.

To show global asymptotic stability, we introduce the Lyapunov candidate $V(y_t) = y_t^\top y_t$ and study if the condition $\dot{V}(y_t) < 0$ holds for all $y_t \in \mathbb{R}^n$. By introducing $A = \text{diag}(-\alpha(y_t))$ and, without loss of generality, setting $y_g = 0$, we write (2) as $\dot{y}_t = Ay_t$. Then

$$\begin{aligned}\dot{V}(y_t) &= (Ay_t)^\top y_t + y_t^\top (Ay_t) \\ &= y_t^\top (A + A^\top) y_t \\ &= 2y_t^\top A y_t \qquad \text{(since A diagonal)}.\end{aligned}$$

Therefore, it follows that this function is negative when the eigenvalues of A are negative. Since A is a diagonal matrix, the eigenvalues correspond to its diagonal $-\alpha(y_t)$. Consequently, y_g is globally asymptotically stable in system (2) when $\alpha(y_t) > 0$.

[12]In discrete time, the integral transforms into a summation.

Appendix C
Neural Network Architecture

In this appendix, we provide details regarding the neural network's architecture. The criteria employed to design the architecture were to build a network: 1) large enough for it to be very flexible in terms of the motions that it can represent, and 2) with a reasonable size such that it can do inference in real time. Consequently, we observed that three feedforward fully connected layers, with 300 neurons each, for ψ_θ and ϕ_θ, i.e., 6 layers in total, were enough for obtaining accurate results and low inference times. The employed activation function was GELU [51] for every layer except for the last layer of the network, which had a linear activation, and for the last layer of α, which had a sigmoid function. In our case, the network inferred at 677 ± 57 Hz (confidence interval with one standard deviation) using a laptop PC with an Intel i7-8750H (12) @ 4.100 GHz CPU and an NVIDIA GeForce RTX 2070 Mobile GPU. PyTorch had the GPU enabled at inference time.

For the case of the adaptive gains $\alpha(y_t^{\mathcal{T}})$, two layers were employed instead. Note that these layers only affect the training time of the network, given that they are not required for inference.

Finally, layer normalization [52] was added after each layer of the network except for the last layers of ψ_θ, ϕ_θ, and α. This type of normalization has shown to be beneficial for reducing training times and also for helping with vanishing gradients.

Appendix D
Hyperparameter Optimization

We introduce a hyperparameter optimization strategy for CONDOR's different variations on the LASA and LAIR datasets. We define an accuracy metric, $\mathcal{L}_{\text{acc}}$, calculated using the distance metrics from Section V-A2. We also evaluate the stability of the system by minimizing the diffeomorphism mismatch, i.e., the RMSE between $y_{1:N}^{\mathcal{L}}$ and $y_{1:N}^{\mathcal{T}}$, defining the stability term, $\mathcal{L}_{\text{stable}}$. Lastly, we account for the precision of the learned system's goal versus the real goal by measuring the average distance of all final trajectory points to the goal, creating the term $\mathcal{L}_{\text{goal}}$. Then, we define the following objective:

$$\mathcal{L}_{\text{hyper}} = \mathcal{L}_{\text{acc}} + \gamma_{\text{stable}}\mathcal{L}_{\text{stable}} + \gamma_{\text{goal}}\mathcal{L}_{\text{goal}}$$

where γ_{stable} and γ_{goal} are weighting factors. After initial tests, we settled on $\gamma_{\text{stable}} = 0.48$ and $\gamma_{\text{goal}} = 3.5$.

In practical applications, hyperparameter tuning has limitations like time consumption and susceptibility to the curse of dimensionality. To mitigate this, we focused on the following five strategies: reducing the objective function's overhead, limiting the evaluation set, employing Bayesian optimization, pruning, and selecting a subset of hyperparameters.

1) Reduced Objective Function's Overhead: The objective function minimized in the hyperparameter optimization process is periodically computed throughout each learning process. Consequently, if this function is expensive to compute, it will make the optimization process slower. More specifically, we observe that the computation of the accuracy using the DTWD and FD metrics adds considerable overhead to the computation time of the objective function. Furthermore, we also observe that the values of the RMSE, DTWD, and FD are highly correlated. Therefore, since computing the RMSE is much faster than computing the other metrics, the hyperparameter optimization loss that accounts for accuracy only consists of the RMSE, i.e., ℓ_{IL} with $H^i = n$ and $t' = 0$.

2) Reduced Evaluation Set: Optimizing hyperparameters for the LASA/LAIR dataset using different motions simultaneously is challenging since the objective computed from different motions is not comparable. Instead, we focused on optimizing using a single, *difficult* motion, assuming robust hyperparameters for it would perform well overall. We selected the *heee* motion from the LASA dataset for first-order motions and the *capricorn* motion from the LAIR dataset for second-order motions. These motions, with complex features like large curvatures or sharp edges, represented challenging test cases.

3) Bayesian Optimization: During optimization, every evaluation of a different set of hyperparameters is costly. Therefore, instead of randomly selecting the hyperparameters to evaluate at each run or following a grid search approach, we select the most promising set given the ones evaluated so far. To achieve this, we employ the Tree Parzen Estimator (TPE) [53], which builds a probability model of the objective function and uses it to select the next set of hyperparameters based on how promising they are. We use the implementation available in the Optuna API [54].

4) Pruning: Throughout the optimization process, it is possible to detect inauspicious runs after a few evaluations. Hence, these trials can be *pruned* before the training process ends, freeing the computational resources for a new run to be executed. We also incorporate this feature in the optimization process using the pruning method available in the Optuna API.

5) Select a Subset of Hyperparameters: Finally, before starting the optimization process, by interacting with CONDOR, we identified a subset of the hyperparameters that showed to have the largest influence over its results. Therefore, to reduce the dimensionality of the search problem, only this subset of hyperparameters is optimized. The rest are manually tuned based on our interactions with the framework.

A. Results

Table IV details the results of the hyperparameters optimization process, including the optimized parameters, and their pre- and postoptimization values. It is divided into two sections: hyperparameters specific to CONDOR, and those general to DNNs. Note that most optimized hyperparameters pertain to the CONDOR method. For the LAIR dataset, we used LASA's optimized hyperparameters as a starting point, resulting in no improved set found for the second-order CONDOR method. Hyperparameters used in BC are excluded as those applicable were identical to those of CONDOR.

Note that the hyperparameter $\alpha_{\max} \in (0, 1]$ has not been introduced yet. This hyperparameter limits the maximum value

TABLE IV
HYPERPARAMETER OPTIMIZATION RESULTS OF CONDOR

Hyperparameter	**Opt.?**	**Hand-tuned value / initial opt. guess**				**Optimized value**			
		CONDOR	**CONDOR (fixed gains)**	**CONDOR (triplet)**	**CONDOR (first order LAIR)**	**CONDOR*** **	**CONDOR (fixed gains)**	**CONDOR (triplet)**	**CONDOR (1st order LAIR)**
CONDOR									
Max adap. latent gain ($\alpha_{\max}$)	✓	1e-2	8e-3*	1e-2	9.997e-2	9.997e-2	2.470e-3*	3.970e-2	0.174
Stability loss weight (λ)	✓	1	1	1	9.300e-2	9.300e-2	3.481	0.280	2.633
Window size imitation (H^i)	✓	14	14	14	14	14	14	14	1
Window size stability (H^s)	✓	4	4	2	4	1	1	2	8
Contrastive margin (m)	✓	1e-2	1e-2	1e-4**	3.334e-2	3.334e-2	3.215e-3	1.977e-4**	1.557e-2
Batch size imitation (B^i)	✗	250	250	250	250	-	-	-	-
Batch size stability (B^s)	✗	250	250	250	250	-	-	-	-
Neural Network									
Optimizer	✗	AdamW	AdamW	AdamW	AdamW	-	-	-	-
Number of iterations	✗	40000	40000	40000	40000	-	-	-	-
Learning rate	✓	1e-4	1e-4	1e-4	1e-4	4.855e-4	4.295e-4	8.057e-4	5.553e-5
Weight decay	✗	1e-4	1e-4	1e-4	1e-4	-	-	-	-
Activation function	✗	GELU	GELU	GELU	GELU	-	-	-	-
Num. layers (ψ_θ, ϕ_θ, α)	✗	(3, 3, 3)	(3, 3, -)	(3, 3, 3)	(3, 3, 3)	-	-	-	-
Neurons/hidden layer	✗	300	300	300	300	-	-	-	-
Layer normalization	✗	yes	yes	yes	yes	-	-	-	-

*Corresponds to the optimized fixed gain.
**Corresponds to the triplet loss margin.
***Also applies for CONDOR (second order).

of the adaptive gain α in $f^{\mathcal{L}}$ (see Section IV-E1). Hence, if in this work we define $\Delta t = 1$ for α in $f^{\mathcal{L}}$; then, even though its maximum allowed value is 1, we limit it even further using $\alpha_{\max}$. This process improves CONDOR's performance, as observed in preliminary experiments. Hence, we define $\alpha = \alpha_{\max}\bar{\alpha}(y_t^{\mathcal{T}})$, with $\bar{\alpha}$ as the DNN output using the sigmoid activation function, ensuring $\alpha \in (0, \alpha_{\max})$.

APPENDIX E
REAL-WORLD EXPERIMENTS: LOW-LEVEL CONTROL

Regarding the low-level control strategy employed in this work, we focus on fully actuated rigid body dynamics systems, which evolve according to the following equation of motion:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) + D(q)\dot{q} = u$$

where q is the joint angle vector, $M(q)$ is the inertia matrix, $C(q,\dot{q})$ is the Coriolis/centripetal vector, $G(q)$ is the gravity vector, $D(q)$ is the viscous friction matrix, and u is the actuation torque vector [55].

The objective of the low-level controller is to, at every time step, map the desired state x^d and state derivative $\dot{x}^d$ provided by CONDOR to u, such that the system is driven toward the desired state. In our experiments, we learn motions in task space and in joint space. Hence, we employ slightly different strategies for each case.

The control frequency of the low-level controller is *500 Hz* in every experiment.

A. Joint Space Control

In this work, independently of the task that CONDOR controls, every motion reference is eventually mapped to joint space. Hence, this section explains our approach to track this reference in joint space $(q^d, \dot{q}^d)$. We achieve this by means of a proportional-derivative controller with gravity compensation, i.e.,

$$u = \alpha(q^d - q) + \beta(\dot{q}^d - \dot{q}) + G(q) \tag{7}$$

where α and β are gain matrices. The higher the gains of this controller, the smaller the tracking error [56], [57]. Moreover, an interesting property of this approach is that as q^d approaches q_g, where q_g corresponds to the mapping from x_g to the configuration space of the robot, CONDOR makes $\dot{x}^d$, and in consequence $\dot{q}^d$, tend to 0. Then, (7) behaves similarly to

$$u = \alpha(q^d - q) - \beta\dot{q} + G(q).$$

This control law ensures global asymptotic stability at the equilibrium q^d for any choice of α and β as long as these are positive definite matrices [55].

B. Task Space Control: Online

To control the robot when references $\dot{x}^d$ are given online [see Fig. 5(a)] in task space, we use the real-time inverse kinematics (IK) library TRACK-IK [58]. To do so, we integrate the velocity reference using the forward Euler integrator to obtain x^d, and we map this position to joint space using this library to obtain q^d. We apply exponential smoothing to these results to alleviate vibrations and stuttering issues. Finally, we compute the desired velocity $\dot{q}^d$ using the forward difference of q, i.e., $\dot{q}^d = (q^d - q)/\Delta t$, where Δt is the time step length of CONDOR. Then, at every time step, the values $q^d, \dot{q}^d$ are provided to the controller described in Appendix E-A.

C. Task Space Control: Offline

In the offline case [see Fig. 5(b)], a trajectory in task space $(x_0^d, x_1^d, \ldots, x_N^d)$ is first computed with CONDOR. This trajectory is fed to the low-level controller to execute it offline. To achieve this, firstly, we map the trajectory to joint space using the Levenberg–Marquadt IK solver of the Robotics Toolbox [59]. In

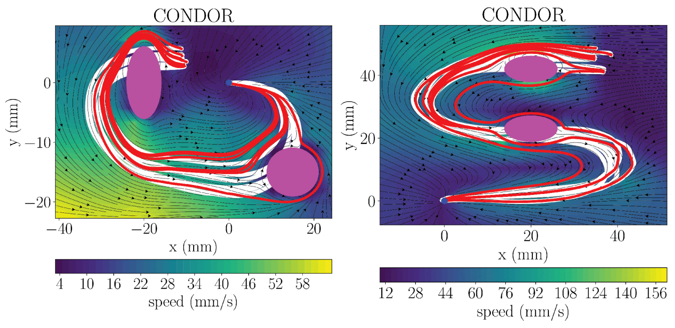

Fig. 19. Obstacle avoidance in the LASA dataset.

this case, we employ this solver instead of TRACK-IK, because it is robust around singularities and can avoid problems like stuttering [60]. This is important for obtaining very smooth solutions in scenarios where this is critical, such as in writing tasks. Note that the solver does not run in real time; however, this is not problematic, since the IK solutions are computed offline.

Afterward, the resulting joint space reference trajectory is approximated with a spline [61] that evolves as a function of time. Finally, when the motion starts, the time is incremented by Δt at each time step and used to query the reference value that is sent to the controller described in Appendix E-A.

Appendix F
Obstacle Avoidance

Obstacle avoidance for motions modeled as dynamical systems is a problem that has been addressed in the literature [44], [46], [62]. Any of these methods can be combined with our proposed framework. In this work, we implemented the method presented in [44] in PyTorch, and combined it with CONDOR. We compute a modulation matrix $M(x)$ that, when multiplied with the learned dynamical system $f(x)$, modifies the motion such that a new dynamical system $\bar{f}(x) = M(x)f(x)$ is obtained. $\bar{f}(x)$ avoids obstacles while maintaining the stability properties of $f(x)$. For more details please refer to [44] (obstacle avoidance of multiple convex obstacles).

Fig. 19 shows motions from the LASA dataset where we test this approach. We observe that the motions generated with CONDOR remain stable after applying the modulation matrix. Furthermore, the obstacles are successfully avoided, showing that, as expected, the dynamical system motion formulation of CONDOR can be effectively combined with methods designed to work with dynamical systems.

Finally, this method was tested with 3-D obstacles in a real 7-DoF robot manipulator when controlling its end effector position. These results are provided in the attached video.

References

[1] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural Comput.*, vol. 25, no. 2, pp. 328–373, 2013.
[2] S. M. Khansari-Zadeh and A. Billard, "Learning stable nonlinear dynamical systems with Gaussian mixture models," *IEEE Trans. Robot.*, vol. 27, no. 5, pp. 943–957, Oct. 2011.
[3] M. A. Rana, A. Li, D. Fox, B. Boots, F. Ramos, and N. Ratliff, "Euclideanizing flows: Diffeomorphic reduction for learning stable dynamical systems," in *Proc. 2nd Annu. Conf. Learn. Dyn. Control*, 2020, pp. 630–639.
[4] J. Kober, M. Glisson, and M. Mistry, "Playing catch and juggling with a humanoid robot," in *Proc. IEEE-RAS 12th Int. Conf. Humanoid Robots*, 2012, pp. 875–881.
[5] V. Sindhwani, S. Tu, and M. Khansari, "Learning contracting vector fields for stable imitation learning," 2018, *arXiv:1804.04878*.
[6] N. Perrin and P. Schlehuber-Caissier, "Fast diffeomorphic matching to learn globally asymptotically stable nonlinear dynamical systems," *Syst. Control Lett.*, vol. 96, pp. 51–59, 2016.
[7] J. Urain, M. Ginesi, D. Tateo, and J. Peters, "ImitationFlow: Learning deep stable stochastic dynamic systems by normalizing flows," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 5231–5237.
[8] S. M. Khansari-Zadeh and A. Billard, "Learning control Lyapunov function to ensure stability of dynamical system-based robot reaching motions," *Robot. Auton. Syst.*, vol. 62, no. 6, pp. 752–765, 2014.
[9] A. Lemme, K. Neumann, R. F. Reinhart, and J. J. Steil, "Neural learning of vector fields for encoding stable dynamical systems," *Neurocomputing*, vol. 141, pp. 3–14, 2014.
[10] M. Awad and R. Khanna, *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers*. Berlin, Germany:Springer, 2015.
[11] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2005, vol. 1, pp. 539–546.
[12] M. Kaya and H. Ş. Bilge, "Deep metric learning: A survey," *Symmetry*, vol. 11, no. 9, 2019, Art. no. 1066.
[13] G. Li, Z. Jin, M. Volpp, F. Otto, R. Lioutikov, and G. Neumann, "ProDMP: A unified perspective on dynamic and probabilistic movement primitives," *IEEE Robot. Automat. Lett.*, vol. 8, no. 4, pp. 2325–2332, 2023, .
[14] H. B. Amor, G. Neumann, S. Kamthe, O. Kroemer, and J. Peters, "Interaction primitives for human-robot cooperation tasks," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014, pp. 2831–2837.
[15] A. Pervez, Y. Mao, and D. Lee, "Learning deep movement primitives using convolutional neural networks," in *Proc. IEEE-RAS 17th Int. Conf. Humanoid Robot.*, 2017, pp. 191–197.
[16] B. Ridge et al., "Training of deep neural networks for the generation of dynamic movement primitives," *Neural Netw.*, vol. 127, pp. 121–131, 2020.
[17] S. Bahl, M. Mukadam, A. Gupta, and D. Pathak, "Neural dynamic policies for end-to-end sensorimotor learning," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, 2020, vol. 33, pp. 5058–5069.
[18] S. Calinon, A. Pistillo, and D. G. Caldwell, "Encoding the time and space constraints of a task in explicit-duration hidden Markov model," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2011, pp. 3413–3418.
[19] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard, "Recent advances in robot learning from demonstration," *Annu. Rev. Control, Robot., Auton. Syst.*, vol. 3, pp. 297–330, 2020.
[20] J. Duan, Y. Ou, J. Hu, Z. Wang, S. Jin, and C. Xu, "Fast and stable learning of dynamical systems based on extreme learning machine," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 49, no. 6, pp. 1175–1185, Jun. 2019.
[21] N. Figueroa and A. Billard, "A physically-consistent Bayesian non-parametric mixture model for dynamical system learning," in *Proc. Conf. Robot Learn.*, 2018, pp. 927–946.
[22] H. Ravichandar, I. Salehi, and A. Dani, "Learning partially contracting dynamical systems from demonstrations," in *Proc. Conf. Robot Learn.*, 2017, pp. 369–378.
[23] K. Neumann and J. J. Steil, "Learning robot motions with stable dynamical systems under diffeomorphic transformations," *Robot. Auton. Syst.*, vol. 70, pp. 1–15, 2015.
[24] H. Ravichandar and A. Dani, "Learning contracting nonlinear dynamics from human demonstration for robot motion planning," in *Proc. Dyn. Syst. Control Conf.*, 2015, vol. 57250, Art. no. V002T27A008.
[25] C. Blocher, M. Saveriano, and D. Lee, "Learning stable dynamical systems using contraction theory," in *Proc. IEEE 14th Int. Conf. Ubiquitous Robots Ambient Intell.*, 2017, pp. 124–129.
[26] C. M. Bishop, "Pattern recognition," *Mach. Learn.*, vol. 128, no. 9, pp. 55–57, 2006.
[27] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 815–823.
[28] H. K. Khalil, *Nonlinear Control*, vol. 406. New York, NY, USA:Pearson, 2015.

[29] J. K. Hunter, "Introduction to dynamical systems," *UCDavis Math. MAT A*, vol. 207, 2011, Art. no. 2011.
[30] J. Abramson et al., "Imitating interactive intelligence," 2020, *arXiv:2012.05672*.
[31] C. Legaard et al., "Constructing neural network based models for simulating dynamical systems," *ACM Comput. Surveys*, vol. 55, no. 11, pp. 1–34, 2023.
[32] T. Osa et al., "An algorithmic perspective on imitation learning," *Found. Trends Robot.*, vol. 7, no. 1/2, pp. 1–179, 2018.
[33] N. Lambert, K. Pister, and R. Calandra, "Investigating compounding prediction errors in learned dynamics models," 2022, *arXiv:2203.09637*.
[34] G. Bontempi, S. Ben Taieb, and Y.-A. L. Borgne, "Machine learning strategies for time series forecasting," in *European Business Intelligence Summer School*. Berlin, Germany: Springer, 2012, pp. 62–77.
[35] A. Venkatraman, M. Hebert, and J. A. Bagnell, "Improving multi-step prediction of learned time series models," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 3024–3030.
[36] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct. 1990.
[37] J. Langford, R. Salakhutdinov, and T. Zhang, "Learning nonlinear dynamic models," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 593–600.
[38] A. Graves, "Generating sequences with recurrent neural networks," 2013, *arXiv:1308.0850*.
[39] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994.
[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[41] A. Veit, M. J. Wilber, and S. Belongie, "Residual networks behave like ensembles of relatively shallow networks," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, 2016, vol. 29, pp. 550–558.
[42] M. Müller, "Dynamic time warping," in *Information Retrieval for Music and Motion*, Berlin, Germany: Springer, 2007, pp. 69–84.
[43] T. Eiter and H. Mannila, "Computing discrete Fréchet distance," Technische Universitat Wien, Tech. Rep. CD-TR 94/64, 1994.
[44] S. M. Khansari-Zadeh and A. Billard, "A dynamical system approach to realtime obstacle avoidance," *Auton. Robots*, vol. 32, no. 4, pp. 433–454, May 2012. [Online]. Available: https://doi.org/10.1007/s10514-012-9287-y
[45] M. Saveriano and D. Lee, "Distance based dynamical system modulation for reactive avoidance of moving obstacles," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014, pp. 5618–5623.
[46] L. Huber, A. Billard, and J.-J. Slotine, "Avoidance of convex and concave obstacles with convergence ensured through contraction," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 1462–1469, Apr. 2019.
[47] L. Huber, J.-J. Slotine, and A. Billard, "Avoiding dense and dynamic obstacles in enclosed spaces: Application to moving in crowds," *IEEE Trans. Robot.*, vol. 38, no. 5, pp. 3113–3132, Oct. 2022.
[48] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, no. 11, pp. 2579–2605, 2008.
[49] A. Ude, B. Nemec, T. Petrić, and J. Morimoto, "Orientation in Cartesian space dynamic movement primitives," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014, pp. 2997–3004.
[50] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. 27th Int. Conf. Mach. Learn.*, 2010, pp. 807–814.
[51] D. Hendrycks and K. Gimpel, "Gaussian error linear units (GELUs)," 2016, *arXiv:1606.08415*.
[52] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016, *arXiv:1607.06450*.
[53] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proc. 24th Int. Conf. Adv. Neural Inf. Process. Syst.*, 2011, vol. 24, pp. 2546–2554.
[54] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 2623–2631.
[55] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*, Force Control, London, U.K.: Springer, 2009, pp. 363–405.
[56] S. Kawamura, F. Miyazaki, and S. Arimoto, "Is a local linear PD feedback control law effective for trajectory tracking of robot motion?," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1988, pp. 1335–1340.
[57] C. Della Santina, C. Duriez, and D. Rus, "Model-based control of soft robots: A survey of the state of the art and open challenges," *IEEE Cont. Syst. Mag.*, vol. 43, no. 3, pp. 30–65, 2023.
[58] P. Beeson and B. Ames, "TRAC-IK: An open-source library for improved solving of generic inverse kinematics," in *Proc. IEEE-RAS 15th Int. Conf. Humanoid Robots*, 2015, pp. 928–935.
[59] P. Corke and J. Haviland, "Not your grandmother's toolbox—The robotics toolbox reinvented for python," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2021, pp. 11357–11363.
[60] I. B. Love, "Levenberg-Marquardt algorithm in robotic controls," Ph.D. dissertation, Dept. Math., University of Washington, Seattle, WA, USA, 2020.
[61] P. Dierckx, *Curve and Surface Fitting With Splines*. London, U.K.:Oxford Univ. Press, 1995.
[62] M. Saveriano and D. Lee, "Point cloud based dynamical system modulation for reactive avoidance of convex and concave obstacles," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2013, pp. 5380–5387.

Rodrigo Pérez-Dattari (Member, IEEE) received the B.Sc. and M.Sc. degrees in electrical engineering from the University of Chile, Santiago, Chile, in 2017 and 2019, respectively. He is currently working toward the Ph.D. degree in robotics with the Delft University of Technology, Delft, The Netherlands.

During his time at the University of Chile, he was a member of the UChile Robotics Team, where he participated in the RoboCup competition in the years 2016–2018. He is a part of the FlexCRAFT project, a Dutch research program to advance robotics for flexible agro-food technology. His research interests include imitation learning, interactive learning, and machine learning for control.

Jens Kober (Senior Member, IEEE) received the Ph.D. degree in engineering from TU Darmstadt, Darmstadt, Germany, and the MPI for Intelligent Systems, Tübingen, Germany, in 2012.

He is currently an Associate Professor with the TU Delft, Delft, The Netherlands. He was a Postdoctoral Scholar jointly with the CoR-Lab, Bielefeld University, Bielefeld, Germany, and with the Honda Research Institute Europe, Offenbach, Germany. His research interests include motor skill learning, (deep) reinforcement learning, imitation learning, interactive learning, and machine learning for control.

Dr. Kober was the recipient of the annually awarded Georges Giralt Ph.D. Award for the best Ph.D. thesis in robotics in Europe, the 2018 IEEE RAS Early Academic Career Award, the 2022 RSS Early Career Award, and was also the recipient of an ERC Starting grant, for his research.