Sensorimotor Learning with Stability Guarantees via Autonomous Neural Dynamic Policies

Dionis Totsila[1*], Konstantinos Chatzilygeroudis[2*],
Valerio Modugno[3], Denis Hadjivelichkov[3] and Dimitrios Kanoulas[3]

***Abstract*— State-of-the-art sensorimotor learning algorithms, either in the context of reinforcement learning or imitation learning, offer policies that can often produce unstable behaviors, damaging the robot and/or the environment. Moreover, it is very difficult to interpret the optimized controller and analyze its behavior and/or performance. Traditional robot learning, on the contrary, relies on dynamical system-based policies that can be analyzed for stability/safety. Such policies, however, are neither flexible nor generic and usually work only with proprioceptive sensor states. In this work, we bridge the gap between generic neural network policies and dynamical system-based policies, and we introduce Autonomous Neural Dynamic Policies (ANDPs) that: (a) are based on autonomous dynamical systems, (b) always produce asymptotically stable behaviors, and (c) are more flexible than traditional stable dynamical system-based policies. ANDPs are fully differentiable, flexible generic-policies that accept any observation input, while ensuring asymptotic stability. Through several experiments, we explore the flexibility and capacity of ANDPs in several imitation learning tasks including experiments with image observations. The results show that ANDPs combine the benefits of both neural network-based and dynamical system-based methods.**

I. Introduction

Choosing the appropriate policy structure is crucial for effective and practical robot learning [1], [2], [3]. Currently, either in the context of Reinforcement Learning (RL) [4] or Imitation Learning (IL) [5], the standard choice is to use Neural Networks (NNs). NNs possess the flexibility needed to learn complicated behaviors as well as the generalizability required to be able to run the same algorithms in any robot or scenario. NN-based policies, however, are black-box and cannot guarantee well-behaved trajectories. In other terms, we cannot anticipate how a NN-based policy will behave when faced with an unforeseen situation. As a consequence, most RL algorithms often attempt behaviors that are harmful to the robot and/or the environment, especially in the initial stages of the learning process.

*Both authors contributed equally to this research.

[1] D. Totsila is with Inria, CNRS, Loria and Université de Lorraine, France. dionis.totsila@inria.fr

[2]K. Chatzilygeroudis is with the Computational Intelligence Laboratory (CILab), Department of Mathematics, University of Patras and Laboratory of Automation and Robotics (LAR), Department of Electrical and Computer Engineering. University of Patras, Greece. costashatz@upatras.gr

[3] V. Modugno, D. Hadjivelichkov and D. Kanoulas are with Robot Perception and Learning Lab (RPL Lab), Department of Computer Science, University College London (UCL), United Kingdom. D. Kanoulas is also with Archimedes/Athena RC, Greece. {v.modugno, dennis.hadjivelichkov, d.kanoulas}@ucl.ac.uk

Code and videos are available at: https://nosalro.github.io/andps

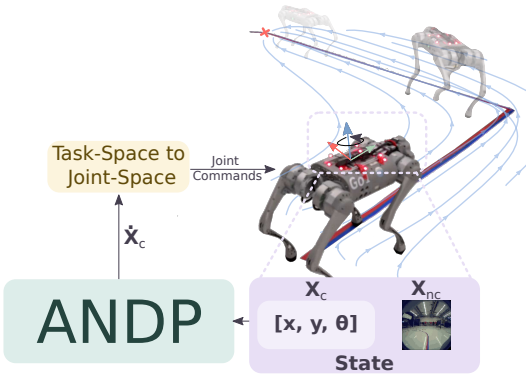

Fig. 1. Autonomous Neural Dynamic Policy Outline.

This comes in contrast with traditional robot learning literature [6], where usually in the context of IL/Learning from Demonstrations (LfD) the policy behavior is shaped according to some well-defined criteria. For example, producing behaviors that are guaranteed to asymptotically converge to an attractor is an important concept of traditional robot learning [6], [7]. The main building tool for this is a Dynamical System (DS), and the main idea is to represent the policy to be learned as a DS. This gives us the ability to reason about the policy in terms familiar to control theory and make proofs about properties that we care about (e.g., for asymptotic stability) [7]. This concept has been explored in robotic scenarios with two main different approaches: (a) time-dependent DSs that mostly fall under the framework of Dynamic Movement Primitives (DMPs) [8], [9], [10], and (b) autonomous DSs where the input is only dependent on the current state [11], [7]. Both approaches can provide asymptotic stability guarantees, while autonomous DSs' reactiveness does not depend on time. Traditionally, both approaches have been utilized mainly in Imitation Learning (IL)/Learning from Demonstrations (LfD) scenarios [12], [13], [14], [15], [7], [16], but recently there was an attempt to use DMPs with RL [17], [18].

For the context of this work, the "ideal" policy representation for effective robot learning: **1)** Produces behaviors that are guaranteed to asymptotically converge to an attractor; **2)** Is general purpose, i.e. it can be used with any robot/embodied agent; **3)** Can accept any observation space (e.g. RGB images), and the input is not limited only to the feedback of robot states. The first characteristic is important for robotic applications since it ensures that the policy outside of the trained data will not perform unstable and unpredictable behaviors, but will try to go towards the attractor. Even if this feature cannot guarantee that

the robot will never try anything harmful (e.g., very big velocities), it is a good stepping stone towards having safe robot behaviors. The second characteristic is important as it will make the policy usable by any robotic mechanism and any scenario. The last characteristic is vital for practical robotic applications since traditionally policies that provide stability guarantees are not straightforward to use with states different from the proprioceptive sensor states.

To create a policy that has the above characteristics we take inspiration from the LfD literature and the usage of DSs, attempt to combine them with the representation abilities of NNs, and: **1)** We assume that the state of the system can be split into two parts: (a) a part that is directly controlled (e.g., joint positions), and (b) a part that can only be observed and/or indirectly controlled (e.g., a box); **2)** We represent the policy as a non-linear combination of linear systems, allowing us to ensure stability while leveraging neural networks to effectively combine the elementary dynamical systems. Our novel type of policies is called **Autonomous Neural Dynamic Policies (ANDPs)** and they: (a) are based on dynamical systems, (b) are generic policies, (c) are using neural networks, (d) do not depend on time (thus, autonomous), and (e) *can learn from a handful of demonstrations*. The main novelty of ANDPs lies in their ability to accept any input space (even images), while still being able to produce **asymptotically stable behaviors for the controllable state**. This is achieved by imposing constraints on the elementary linear DSs while making the policy expressive by using NNs to combine them. Finally, we perform reparameterizations to "eliminate" the constraints, and optimize ANDPs using unconstrained gradient-based optimization.

II. Related Work

The choice of the policy structure plays an important role in the effectiveness of learning in practical robot applications. When designing the policy structure, there is always a trade-off between having a representation that is expressive, and one that provides a space that is efficiently searchable [2].

A straightforward approach to ensure the policy is easily found and defined is to design it manually. For example, In [19], the authors design a policy for a ball acquisition task by hand, which has only four parameters. This low-dimensional policy can be easily optimized, but with only four parameters it might not possess big expressiveness. Moreover, if we are faced with a different robot/task, we need to re-design the policy from scratch. On the other hand, using a function approximator (e.g. a neural network) to describe the policy, enables us to easily increase the expressiveness of the policy, but can make the policy optimization difficult.

Numerous works describe best practices and policy structures that ease the learning process. In [3] the authors thoroughly evaluate different action spaces (with the corresponding low-level controllers, and thus policy structures) in a wide range of scenarios. They conclude that operating in end-effector space combined with low-level controllers makes the learning process faster, more robust, and provides easier transfer from simulation to the physical world and between robots. The authors in [1] reach similar conclusions in a related study that performs a comparison between different action spaces for manipulation tasks. Overall, it is clear that in order to learn effectively in practical robotic applications, we need a structured policy representation.

Dynamic Movement Primitives (DMPs) [8], [9], [10] provide a framework for structured policy types that are a dynamical system. As such, we can insert desired properties that can make our system behave in specific ways. DMPs are split into two systems: (a) the canonical system (which is usually a springer-damper system), and (b) the transformation system. The canonical system represents the movement *phase* s, which starts at 1 and converges to 0 over time. The transformation systems combine a spring-damper system with a function approximator (e.g., NNs) which, when integrated, generates accelerations. Multi-dimensional DMPs are achieved by coupling multiple transformation systems with one canonical system. DMPs can be used in the end-effector and the joint angle space. There are numerous successful implementations of DMPs mainly in IL/LfD scenarios covering a wide range of tasks [10], [14], [13] and even multi-task cases [12], [20].

DMPs, however, are time-dependent and thus they may produce undesirable behaviors; for example, a policy that cannot adapt to perturbations after some time. Stable Estimator of Dynamical Systems (SEDS) [7] explores how to use dynamical systems to define autonomous (i.e., time-independent) controllers (or policies) that are asymptotically stable. The main idea of the algorithm is to use a finite mixture of Gaussian functions (Gaussian Mixture Models - GMMs) as the policy, $\dot{\boldsymbol{\xi}} = \pi_{\text{seds}}(\boldsymbol{\xi})$, with specific properties that satisfy some stability guarantees. SEDS, however, requires demonstrated data to optimize the policy (i.e., data gathered from experts), although similar ideas have been used within the RL framework [21]. SEDS and its variants [15], [22] have provided effective solutions to difficult tasks ranging from point-to-point motions [7] to humanoid navigation [23] and following force profiles [16]. One of the main limitations of SEDS is the *accuracy vs. stability dilemma*, i.e., it performs poorly in highly non-linear motions that contain high curvatures or that are non-monotonic. This is mainly because of the constraints SEDS imposes on the structure of the GMMs. Recent variants of SEDS, and in particular LPV-DS [15], [22], attempt to relax the constraints by disconnecting the learning of the weighting function from the elementary DSs; LPV-DS still uses GMMs though.

Recently, Bahl et al. [17] proposed a method to combine neural networks with DMPs. The main idea is to create a high-level controller with NNs that takes as input an unstructured state and *selects* parameters of a DMP that acts as the low-level controller. Their method, called Neural Dynamic Policies (NDPs), was able to effectively learn multiple LfD and RL scenarios. In a recent extension [18], the authors provide a hierarchical formulation of their method that can be used to solve more complex tasks. To the best of our knowledge, this work proposes one of the first methods that effectively combines NNs with DSs and provides the

first general-purpose policy (i.e., it can be used with almost any input and any robot) that is based on DSs. However, since their policy changes the dynamical system every X steps there are still no theoretical guarantees for stability, but mostly rely on the data-driven capabilities of the NNs to capture this type of behavior.

In this paper, we take inspiration from LPV-DS and NDPs and provide a policy structure, called Autonomous Neural Dynamic Policies (ANDPs), that (a) always produces asymptotically stable behaviors for the controllable part of the state, and that (b) is a general purpose policy that can accept arbitrary action space and inputs (e.g., images).

III. Problem Formulation

We assume discrete-time dynamical systems that can be described by transition dynamics of the form: $\boldsymbol{x}^{t+1} = f(\boldsymbol{x}^t, \boldsymbol{u}^t) + \boldsymbol{w}$, where the system is at state $\boldsymbol{x}^t \in \mathbb{R}^E$ at time t, takes control input $\boldsymbol{u}^t \in \mathbb{R}^U$ and ends up at state $\boldsymbol{x}^{t+1}$ at time $t+1$, $\boldsymbol{w}$ is i.i.d. Gaussian system noise, and f is a function that describes the unknown transition dynamics. We assume that the system is controlled through a parameterized *policy* $\boldsymbol{u} = \pi(\boldsymbol{x}|\boldsymbol{\theta})$ that is followed for M steps ($\boldsymbol{\theta}$ are the policy parameters). When following a policy for M time-steps from an initial state distribution $p(\boldsymbol{x}^0)$, the system's states and actions jointly form *trajectories* $\boldsymbol{\tau} = (\boldsymbol{x}^0, \boldsymbol{u}^0, \ldots, \boldsymbol{x}^{M-1})$, which are often also called *rollouts*.

In this work, we define a novel policy structure and learning procedure (called ANDPs) with stability guarantees. ANDPs can work in both IL/LfD settings as well as RL scenarios, but in this manuscript we focus on the first type of scenarios. In an imitation learning scenario, we assume access to a few demonstrated trajectories $\{\boldsymbol{\tau}_i\}_{i=1,\ldots,K}$, and we want to find the policy parameterization $\boldsymbol{\theta}$ that "mimics" the demonstrated trajectories as well as possible. In this work, we assume having access only to the states of the system, $\boldsymbol{x}^t$, and not to the control signals, $\boldsymbol{u}^t$. In other words, we have trajectories $\{\boldsymbol{s}_i\}_{i=1,\ldots,K}$ of the form $\boldsymbol{s} = (\boldsymbol{x}^0, \ldots, \boldsymbol{x}^{M-1})$. This makes the problem slightly more difficult and usually enforces the use of a low-level controller [24].

IV. Proposed Policy Structure

We make the assumption that the state of the system can be split into two parts: (a) a part that can be directly controlled (e.g., positions and velocities of the end-effector), and (b) a part that can only be observed and/or indirectly controlled (e.g., obstacles/objects). In particular (we omit the time notation, t, for clarity):

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_c \\ \boldsymbol{x}_{nc} \end{bmatrix} \in \mathbb{R}^{d_c + d_{nc}}, \tag{1}$$

where $\boldsymbol{x}_c$ is the part of the state that can be directly controlled and $\boldsymbol{x}_{nc}$ is the part of the state that can only be observed. d_c and d_{nc} are the state-space dimensions for the controllable and non-controllable parts, respectively ($d_c + d_{nc} = E$). We define the control policy as a dynamical system with a fixed attractor $\boldsymbol{x}_c^*$ (formulated as a weighted sum of linear DSs):

$$\dot{\boldsymbol{x}}_c = \pi(\boldsymbol{x}) = \sum_{i=1}^{N} w_i(\boldsymbol{x}) \boldsymbol{A}_i \Big(\boldsymbol{x}_c^* - \boldsymbol{x}_c \Big) \tag{2}$$

where N denotes the number of elementary dynamical systems, $w_i(\boldsymbol{x}) \in \mathbb{R}$ are state-dependent weighting functions, and $\boldsymbol{A}_i \in \mathbb{R}^{d_c \times d_c}$, $\boldsymbol{x}_c^* \in \mathbb{R}^{d_c}$. The number of the elementary dynamical systems, N, is essentially a hyper-parameter for the policy. It is evident that as the complexity of movement grows, the required number of dynamical systems may increase. However, it is important to note that unlike SEDS, and most of its variants, we do not need to keep increasing N to increase the expressiveness of our policy; we can increase the complexity of the $w_i(\boldsymbol{x})$ functions and keep N small. Although we could employ a principled way of selecting N, the choice of N did not strongly affect the final performance, and we, thus, leave this exploration for future work.

The control policy, $\pi(\boldsymbol{x})$ (Fig. 1), defines the desired velocity profile that the controllable state $\boldsymbol{x}_c$ should follow. Depending on the state representation one can directly use the output for commanding the robot, use a PD controller, or use some inverse dynamics/kinematics model. Note, that the controllable state $\boldsymbol{x}_c$ can also contain velocities (e.g., $\boldsymbol{x}_c = \{\boldsymbol{\xi}, \dot{\boldsymbol{\xi}}\}$, where $\boldsymbol{\xi}$ is the end-effector translation) and in that case the system is a second order DS. Although in this work we explore first-order DSs, our formulation allows for second-order DS systems.

Theorem 1: Assume that the controllable part of a state trajectory follows the policy as defined in Eq. 2. Then, the function described by Eq. 2 is asymptotically stable to $\boldsymbol{x}_c^*$ if

$$\begin{cases} \boldsymbol{A}_i + \boldsymbol{A}_i^T \succ 0 & \text{the symmetric part of } A \text{ is psd} \\ w_i(\boldsymbol{x}) > 0, & i = 1, .., N, \forall \boldsymbol{x} \in \mathbb{R}^E \end{cases} \tag{3}$$

Proof: The dynamical system of Eq. (2) can be re-written as follows:

$$\dot{\boldsymbol{x}}_c = \pi(\boldsymbol{x}) = g(\boldsymbol{x}_c, \boldsymbol{x}_{nc}) \tag{4}$$

where $g(\boldsymbol{x}_c, \boldsymbol{x}_{nc}) = \sum_{i=1}^{N} w_i(\boldsymbol{x}_c, \boldsymbol{x}_{nc}) \boldsymbol{A}_i \Big(\boldsymbol{x}_c^* - \boldsymbol{x}_c \Big)$.

Therefore,to prove the stability of the controllable part of the state, we follow a Lyapunov stability analysis and we will use the Control Lyapunov Function (CLF) theory [25], [26], [27], [28] that states: *A dynamical system that is governed by the dynamics of Eq.* (4)*, with state $\boldsymbol{x}_c$ and control inputs $\boldsymbol{x}_{nc}$, is globally asymptotically stable at the point $\boldsymbol{x}_c^*$ if there exists a continuous and continuously differentiable Lyapunov function $V(\boldsymbol{x}_c) : \mathbb{R}^{d_c} \to \mathbb{R}$ such that:*

$$\begin{cases} \text{(a) } V(\boldsymbol{x}_c) > 0, & \forall \boldsymbol{x}_c \in \mathbb{R}^{d_c}, \boldsymbol{x}_c \neq \boldsymbol{x}_c^* \\ \text{(b) } \exists \boldsymbol{x}_{nc} \text{ s.t. } \dot{V}(\boldsymbol{x}_c, \boldsymbol{x}_{nc}) < 0, & \forall \boldsymbol{x}_c \in \mathbb{R}^{d_c}, \boldsymbol{x}_c \neq \boldsymbol{x}_c^* \\ \text{(c) } V(\boldsymbol{x}_c^*) = 0 & \\ \text{(d) } \dot{V}(\boldsymbol{x}_c^*, \boldsymbol{x}_{nc}) = 0, & \forall \boldsymbol{x}_{nc} \in \mathbb{R}^{d_{nc}} \end{cases} \tag{5}$$

Since in our case $\boldsymbol{x}_{nc}$ is not a control input, but we can just observe it, we go one step further and require Eq. 5b to hold for all $\boldsymbol{x}_{nc}$ (i.e., $\forall \boldsymbol{x}_{nc} \in \mathbb{R}^{d_{nc}}$). If we can find a Lyapunov function that fulfills the above conditions, then the dynamical system defined in Eq. (2) will asymptotically converge to $\boldsymbol{x}_c^*$, that is: $\lim_{t \to \infty} \|\boldsymbol{x}_c - \boldsymbol{x}_c^*\| = 0$.

We propose the following Lyapunov function:

$$V(\boldsymbol{x}_c) = \frac{1}{2}(\boldsymbol{x}_c - \boldsymbol{x}_c^*)^T(\boldsymbol{x}_c - \boldsymbol{x}_c^*). \tag{6}$$

The derivative of the proposed Lyapunov function can be calculated as (the attractor is fixed and thus $\dot{\boldsymbol{x}}_c^* = 0$):

$$\begin{aligned}\frac{dV}{dt} = \dot{V}(\boldsymbol{x}_c, \boldsymbol{x}_{nc}) &= \frac{dV}{d\boldsymbol{x}_c}\frac{d\boldsymbol{x}_c}{dt} + \cancelto{0}{\frac{dV}{d\boldsymbol{x}_{nc}}\frac{d\boldsymbol{x}_{nc}}{dt}} \\ &= \frac{1}{2}\Big(\dot{\boldsymbol{x}}_c^T(\boldsymbol{x}_c - \boldsymbol{x}_c^*) + (\boldsymbol{x}_c - \boldsymbol{x}_c^*)^T\dot{\boldsymbol{x}}_c\Big)\end{aligned} \tag{7}$$

It easy to see that $V(\boldsymbol{x}_c) > 0$ (because of the quadratic form), $V(\boldsymbol{x}_c^*) = 0$ and $\dot{V}(\boldsymbol{x}_c^*, \boldsymbol{x}_{nc}) = 0,\ \forall \boldsymbol{x}_{nc}$. For satisfying Eq. 5b, we substitute Eq. (4) into the above and get:

let $\boldsymbol{e} = \boldsymbol{x}_c - \boldsymbol{x}_c^*$,

$$\begin{aligned}\dot{V}(\boldsymbol{x}_c, \boldsymbol{x}_{nc}) =& \frac{1}{2}\Big((\sum_{i=0}^{N} w_i(\boldsymbol{x}_c, \boldsymbol{x}_{nc})\boldsymbol{A}_i(-\boldsymbol{e})^T\boldsymbol{e} \\ &+ \boldsymbol{e}^T(\sum_{i=0}^{N} w_i(\boldsymbol{x}_c, \boldsymbol{x}_{nc})\boldsymbol{A}_i(-\boldsymbol{e})\Big) \\ =& \frac{1}{2}\Big(\boldsymbol{e}^T\sum_{i=0}^{N} -w_i(\boldsymbol{x}_c, \boldsymbol{x}_{nc})\boldsymbol{A}_i^T\boldsymbol{e} \\ &+ \boldsymbol{e}^T\sum_{i=0}^{N} -w_i(\boldsymbol{x}_c, \boldsymbol{x}_{nc})\boldsymbol{A}_i\boldsymbol{e}\Big) \\ =& \frac{1}{2}\Big(\boldsymbol{e}^T\sum_{i=0}^{N} \underbrace{w_i(\boldsymbol{x}_c, \boldsymbol{x}_{nc})}_{>0}\Big(\underbrace{\boldsymbol{A}_i^T + \boldsymbol{A}_i}_{\succ 0}\Big)\boldsymbol{e}\Big) \\ =& -\frac{1}{2}\Big(\boldsymbol{e}^T\sum_{i=0}^{N} \underbrace{w_i(\boldsymbol{x})}_{>0}\Big(\underbrace{\boldsymbol{A}_i^T + \boldsymbol{A}_i}_{\succ 0}\Big)\boldsymbol{e}\Big) \\ <& 0.\end{aligned} \tag{8}$$

The above holds for $(\boldsymbol{A}_i^T + \boldsymbol{A}_i) \succ 0$ and $w_i(\boldsymbol{x}) > 0,\ i = 1, \ldots, N$. We remind the reader that a positive definite matrix $\boldsymbol{P} \in \mathbb{R}^{n\times n}$ satisfies the condition $\boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x} > 0, \forall \boldsymbol{x} \in \mathbb{R}^n$. ■

The results of the above theorem can be described as *"The controllable part of the system will always converge to the fixed attractor* $\boldsymbol{x}_c^*$*"*. Although this does not guarantee that the whole system state will converge to a desired state, this is an important property for a policy, as it will always generate commands that will eventually drive the controllable part of the system to a stable point. It is important to note that this property holds if the controllable system can follow the commanded velocities perfectly, and does not take into account the properties of a possible low-level controller (e.g., a controller based on the pseudoinverse of the Jacobian for end-effector control) or the rest of the environment. This is, however, common in the LfD literature since designing a policy that can guarantee the stability of the whole system and take into account the properties of the low-level controller is a challenging task and would require bulk approximations to be made [2], [24]. Nevertheless, in all of our experiments, we never observed diverging motions and we did not have to tune the low-level controllers to avoid such situations. Overall, these limitations do not seem to have a big impact on the resulting behaviors and we were able to learn a wide range of motions using different low-level controllers (in joint- and task-space).

It is important to emphasize that the decision to work in Euclidean space for developing stable controllers is arbitrary. Our methodology—splitting the state into controllable and uncontrollable parts and using neural networks to learn weighting functions—remains applicable even when leveraging more advanced mappings, such as those using manifolds [29] or Laplacian Eigenmaps [30].

A. ANDPs via Neural Networks

To be able to represent the ANDPs (Eq. 2) with a neural network, we have to: (a) find the "learnable" parameters, (b) make sure that the policy is fully differentiable, and (c) handle the constraints of Eq. 3 properly.

To define the learnable parameters, we need to identify parameters of Eq. 2. First, the matrices $\boldsymbol{A}_i$ do not depend on the state, and thus we can directly optimize for their parameters. Second, to define each $w_i(\boldsymbol{x})$, we use one neural network for all of them. More concretely, we define a neural network Ψ which takes input a full system state $\boldsymbol{x}$ and predicts a vector $\boldsymbol{W} \in \mathbb{R}^N$. In other words, $\boldsymbol{W} = \Psi(\boldsymbol{x}|\boldsymbol{\psi})$, where $\boldsymbol{\psi}$ are the parameters of the neural network. The i-th element of the $\boldsymbol{W}$ vector represents $w_i(\boldsymbol{x})$. So, the total learnable parameters of the policy are $\boldsymbol{\theta} = \{\boldsymbol{A}_1, \ldots, \boldsymbol{A}_N, \boldsymbol{\psi}\}$. It is easy to see that since each $\boldsymbol{A}_i$ is a simple matrix, and $\boldsymbol{\psi}$ parameters of a neural network, the whole policy is fully differentiable.

One can also add the attractor point, $\boldsymbol{x}_c^*$, to the optimization variables. The policy would still be differentiable ($\boldsymbol{x}_c^*$ is a free parameter). In RL scenarios, adding the attractor point to the optimization variables is mandatory, in order to let the algorithms identify the underlying behavior. In our experiments, however, we observed that in IL/LfD scenarios, it would require a complicated loss function, and thus we left this exploration for future work. In this paper, we assume a fixed attractor that is equal to the average of the last points of all the demonstrated trajectories.

The final component involves optimizing the parameters $\boldsymbol{\theta}$ in accordance with a specific objective function, while satisfying the constraints outlined in Eq. 3. In the general case, this would require a constrained optimization problem to be performed, but this can be challenging to do for high-dimensional parameter spaces, such as the ones generated by neural networks. To bypass this issue but still respect the constraints, we perform the following steps:

- First, each w_i should be positive. We can easily generate positive numbers by adding an exp layer after the last layer of Ψ. In this work, we always use a *softmax* layer as the last layer of Ψ that generates positive values that sum to one; it also makes more sense as we want to combine the elementary DSs that $\boldsymbol{A}_i$ define.
- We, now, need to satisfy the constraint $\boldsymbol{A}_i + \boldsymbol{A}_i^T \succ 0$. If we assume real matrices and define $\boldsymbol{A}_i = \boldsymbol{B}_i + \boldsymbol{C}_i - \boldsymbol{C}_i^T$, where $\boldsymbol{B}_i$ is a symmetric positive definite matrix

(no restrictions for $\boldsymbol{C}_i$), we can see that $\boldsymbol{A}_i + \boldsymbol{A}_i^T = \boldsymbol{B}_i + \boldsymbol{C}_i - \boldsymbol{C}_i^T + \boldsymbol{B}_i + \boldsymbol{C}_i^T - \boldsymbol{C}_i = 2\boldsymbol{B}_i \succ 0$. $\boldsymbol{B}_i$ is symmetric, thus $\boldsymbol{B}_i = \boldsymbol{B}_i^T$, and $\boldsymbol{C}_i - \boldsymbol{C}_i^T$ defines a skew symmetric matrix. This gives us the ability to freely optimize for the parameters of $\boldsymbol{C}_i$. So, we are left with handling the case of optimizing for a symmetric positive definite matrix, $\boldsymbol{B}_i$.

- If we assume real matrices, a symmetric positive definite matrix $\boldsymbol{B}_i$ can be factorized as $\boldsymbol{B}_i = \boldsymbol{L}_i \boldsymbol{L}_i^T$ (Cholesky decomposition), where $\boldsymbol{L}_i$ is a lower triangular matrix with real and positive diagonal entries. It is easy to see that we can optimize for the parameters of $\boldsymbol{L}_i$ and reconstruct $\boldsymbol{B}_i$ that we need.

Using the above steps we can now perform unconstrained optimization while still ensuring that the constraints in Eq. 3 are fulfilled. This is important as we are more confident that the optimization will converge to good solutions, and that the learning process will converge in few epochs/iterations.

B. Training ANDPs for Imitation Learning

Training ANDPs for IL/LfD scenarios is quite straightforward. Given a set of demonstrated trajectories $\{\boldsymbol{s}_i\}_{i=1,\dots,K}$, we create a dataset of the form $\boldsymbol{x} \rightarrow f_{\text{target}}(\boldsymbol{x}) = \dot{\boldsymbol{x}}_c$.

$$\mathcal{L}_{\text{imitation}}(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}} \|\pi(\boldsymbol{x}|\boldsymbol{\theta}) - f_{\text{target}}(\boldsymbol{x})\|^2 \tag{9}$$

Using $\mathcal{L}_{\text{imitation}}$ as the loss function, we can train the parameters $\boldsymbol{\theta}$ of the policy to mimic the behavior of the demonstrated trajectories.

V. Experiments & Results

In this section, we want to show the ability of ANDPS to work in LfD scenarios. Through the conducted experiments, we attempt to answer the following questions:

1) How do ANDPs compare to classical LfD and pure NN approaches? Do ANDPs produce stable behaviors? How well can they reproduce the demonstrations?
2) Can ANDPs learn complex movements for a realistic robotic task? Are they robust?
3) Can ANDPs accept arbitrary inputs like 3D orientations? Can they accept even raw images?
4) Can ANDPs work on a physical robot?

To answer the first question, we use the LASA handwritten open-source dataset [1] that contains multiple 2D movements. The dataset provides a wide range of movements (from simple lines to quite complicated shapes) which allows us to identify the modeling capabilities of different policy structures. The 2D space allows even simple policies to be trained extensively, serving as a useful benchmark for LfD techniques. We provide a quantitative and qualitative analysis of ANDPs as well as an extensive comparison with SEDS [7], and unstructured neural network policies. SEDS and its variants are the main representatives of LfD methods based on autonomous DSs. We do not compare to newer SEDS variants as the main advantage of ANDPs lies in the usage of neural networks instead of GMMs, and many of these improvements to SEDS can be applied to ANDPs analogously. The comparison with NNs serves as a baseline to showcase the need for stability guarantees.

To answer the second and third questions, we devise a multi-task scenario where the goal is to show that ANDPs are capable of learning multiple tasks, for example complex robotic movements, into one policy; we use the RobotDART open-source simulator [31], [32]. The idea is to use the non-controllable part of the state $\boldsymbol{x}_{nc}$ to "define" which task we want the robot to perform. So for each task, $\boldsymbol{x}_{nc}$ is an image captured with a camera that is mounted to the robot's end-effector and points directly to a sign that displays the picture that corresponds to the particular movement. We also test the reactiveness of ANDPs by changing the image in the middle of the evaluation pipeline, and the robustness of the learned policies by inserting force perturbations.

To answer the fourth question, we devise the following 2 experiments: *We use a physical Franka Panda robot to collect three demonstrations with kinesthetic guidance for a pouring task and learn a policy with data collected from a physical setup.* In this task the robot needs to pour liquid from one cup into a bowl and we control the robot in end-effector space with changing orientation. In the subsequent experiment, *a Go1 robot was utilized to navigate an L-shaped trajectory, leveraging both its current positional data and the visual information gathered through the onboard camera*. The goal of this task was to follow as closely as possible the predetermined path, aiming to arrive at the final goal point located at the path's end.

All the videos showcasing the resulting behaviors of the key experiments, as well as the code for replicating the experiments can be found at `https://nosalro.github.io/andps`.

A. Imitating 2D Trajectories (Question 1)

The LASA handwritten dataset contains a set of 2D handwriting motions recorded from a tablet (Fig. 3). For each motion or task, a human was asked to draw a desired pattern 7 times, by starting from different initial positions and ending to the same final point. In total the library contains 30 human handwriting tasks (with 7 demonstrations for each task). The goal of this section is to validate how well ANDPs perform in classical LfD tasks. In other words, given a set of K demonstrations of the same task, how well do ANDPs capture the overall shape and reproduce the original demonstrations? To provide strong evidence that ANDPs perform adequately, we compare to SEDS[2] and unstructured neural networks. Here, the state is of the form $\boldsymbol{x} \equiv \boldsymbol{x}_c = \{x, y\}$, i.e. the state vector contains only the 2D Cartesian position of the system, and we perform two types of evaluation: (a) quantitative, and (b) qualitative.

For (a), we generate 35 distinct dataset splits, where each split consists of 4 demonstrations for the training set and 3 demonstrations for the test set, covering all possible combinations of 4 demos for training and 3 demos for testing. We then average across every trajectory by sliding windows

[1] `https://github.com/justagist/pyLasaDataset`

[2] We used the official open-source code found in `https://bitbucket.org/khansari/seds`.

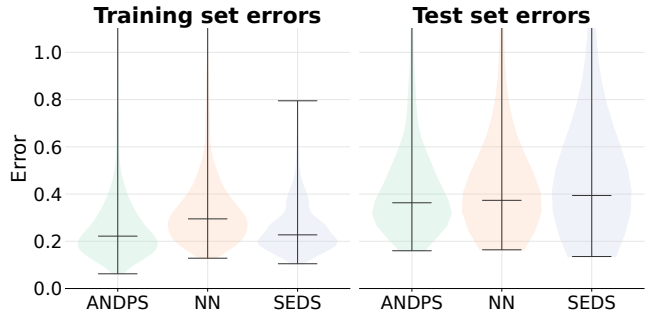

Fig. 2. LASA dataset: Aggregated results over all tasks

of length 5, so that the trajectories are smoother[3], and calculate the corresponding velocities using $\dot{\boldsymbol{x}}_{c_t} = \frac{\boldsymbol{x}_{c_{t+1}} - \boldsymbol{x}_{c_t}}{dt}$. Assuming $\pi(\boldsymbol{x})$ to be the learned model and $\dot{\boldsymbol{x}}_{c\dagger}$ to be the ground truth value, we compute the error as using the formula from [7]. We aggregate the errors from all tasks. The results showcase that ANDPs have comparable (and slightly better) training and test set errors with SEDS and NNs, which indicates that ANDPs can learn complex functions (Fig. 2).

For (b), we select one learned policy for each task and perform a forward simulation for each of the starting positions. In this case we do not provide any feedback, but take only the first position from the dataset and then rely solely on the model predictions (via integration $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{v} * dt$). The results show that ANDPs and SEDS produce behaviors that always converge to the attractor, while NNs cannot do so. ANDPs overall capture the shape of the motion as well as or better than SEDS (Fig. 3).

B. Imitating Robotic Behaviors (Questions 2 & 3)

In this section, we want to determine whether ANDPs have the capability of learning intricate 3D motions, demonstrate the flexibility of ANDPs compared to traditional LfD methods, and exhibit the reactive and resilient nature of the learned policy against perturbations. We collect one demonstration for each of the following movements: a sinusoidal motion, a linear motion, and a curvilinear motion so that we can devise a multi-task scenario where the goal is for ANDPS to learn multiple tasks into one policy. In essence, we use the non-controllable part of the state to "define" which task we want the robot to perform. To create the labels, we simulate a sign with the image corresponding to every motion across the robot, we then attach a camera to the end-effector of the arm facing towards the sign and we shoot a grayscale image for every sample. We take all 3 demonstrations and we have a state of the form: $\boldsymbol{x} = \{\boldsymbol{x}_{nc}, \boldsymbol{x}_c\} = \{\mathcal{I}, x, y\}$, where $\mathcal{I} \in \mathbb{R}^{64\times64}$ is a grayscale image and $\boldsymbol{x}_c$ contains only the 3D Cartesian position of the end-effector and the output is the desired velocity profile $\dot{\boldsymbol{x}}_c$ that the end-effector should follow, while a PID Controller is used to keep the orientation fixed. The robot is controlled through joint commands using the pseudo-inverse of the Jacobian of the end-effector. The joint commands are calculated as: $\dot{\boldsymbol{\theta}}(t) = J^{\dagger}(\boldsymbol{\theta}) \left[\dot{\boldsymbol{x}}_{c_t} \dot{\boldsymbol{\alpha}}(t)\right]^T$ where $\dot{\boldsymbol{\alpha}}(t)$ comes from the PID controller that tries to keep the orientation fixed throughout the trajectory.

We learn one model for all three tasks. In Fig. 4, we see that ANDPs can learn to distinguish the three tasks while always ensuring convergence to the fixed attractor. To highlight the importance of stability guarantees, we train a CNN-based policy and we conduct comparisons for the following perturbations applied during evaluation: a) changes in the non-controllable part of the state, b) external force application, and c) application of i.i.d. Gaussian noise.

To showcase the broader concept of reactiveness we start an evaluation run with the image corresponding to the linear movement displayed, and at $t = 2.5\,s$ we switch the displayed image to the one representing the sinusoidal movement. We observe that when utilizing ANDPs, the robot adapts its motion to align with the shape of the corresponding movement and converges to the fixed attractor. In contrast, while the simple CNN also modifies its motion, the end-effector deviates from the target (Fig. 5). To show that ANDPs are robust and reactive to force perturbations, we apply an external force to the robot twice during the execution: once at the beginning of the behavior, and once at $t = 4\,s$. We observe that the robot can converge to the attractor and follow the overall shape of the behavior, whereas the CNN-based policy fails to do so (Fig. 6).

C. Physical Robot Experiments

1) Pouring Task: In this section, we want to identify whether ANDPs: (a) work with realistic demonstrations, and (b) can learn a task that requires precision and end-effector orientation control. For those reasons, **we collect three demonstrations with kinesthetic guidance on the physical robot performing a pouring task**: the robot holds a cup filled with liquid and needs to pour it inside a bowl. For safety, we "emulate" the liquid with small plastic objects. We then learn a policy with ANDPs using the collected demonstrations with a state of the form $\boldsymbol{x} \equiv \boldsymbol{x}_c = \{x, y, z, r_x, r_y, r_z\}$, where $\{x, y, z\}$ is the end-effector translation and $\{r_x, r_y, r_z\}$ is the end-effector orientation expressed in Euler XYZ angles. To apply joint commands to the physical robots, we convert the EulerXYZ velocities to angular velocities and then the commands are calculated using the pseudo-inverse of the Jacobian of the end-effector. *Here it is important to highlight that we choose to learn the policy in end-effector space to ensure that the learned policy can seamlessly adapt to any robot, enhancing its versatility and usability.* ANDPs can also work on any arbitrary state space, like the joint space, as long as the controllable part of the state forms an Euclidean space.

The results showcase that ANDPs work reliably in this setting and the robot successfully pours the liquid from the cup to the bowl. To validate more thoroughly the effectiveness of the learned policy, we perform *10 replicates with different initial configurations* of the robot and measure *the percentage of the plastic objects that end up inside the bowl*. We get a median percentage of 100% with 67.5% and 100% for the 25-th and 75-th percentiles respectively.

[3]Smoothing was needed for SEDS as it was producing better results with smoothed trajectories; ANDPs were more robust to this noise.

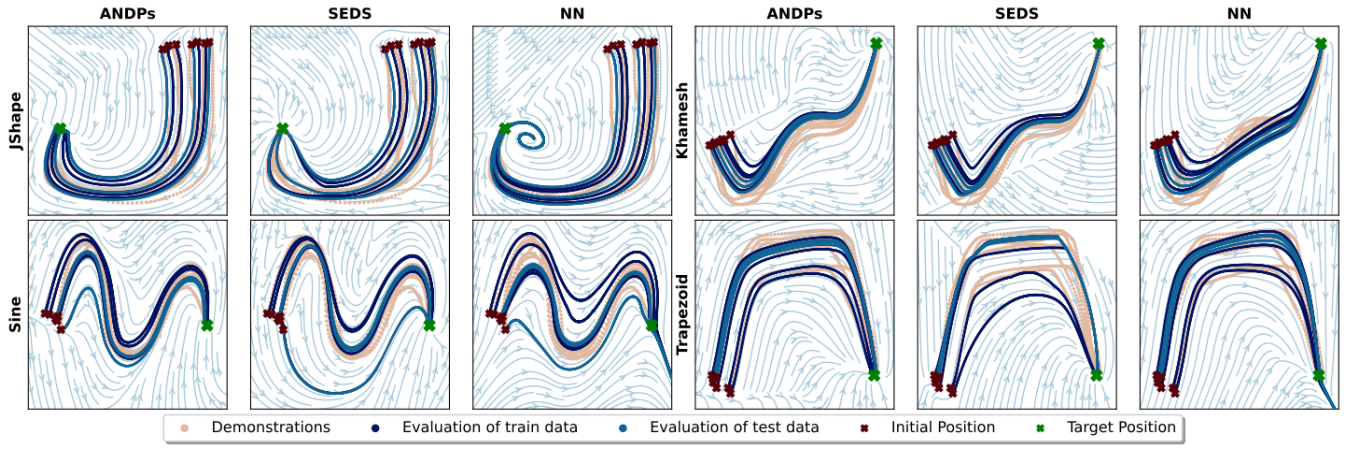

Fig. 3. Qualitative results for ANDPs, SEDS, and NNs in 4 selected tasks of the LASA dataset.

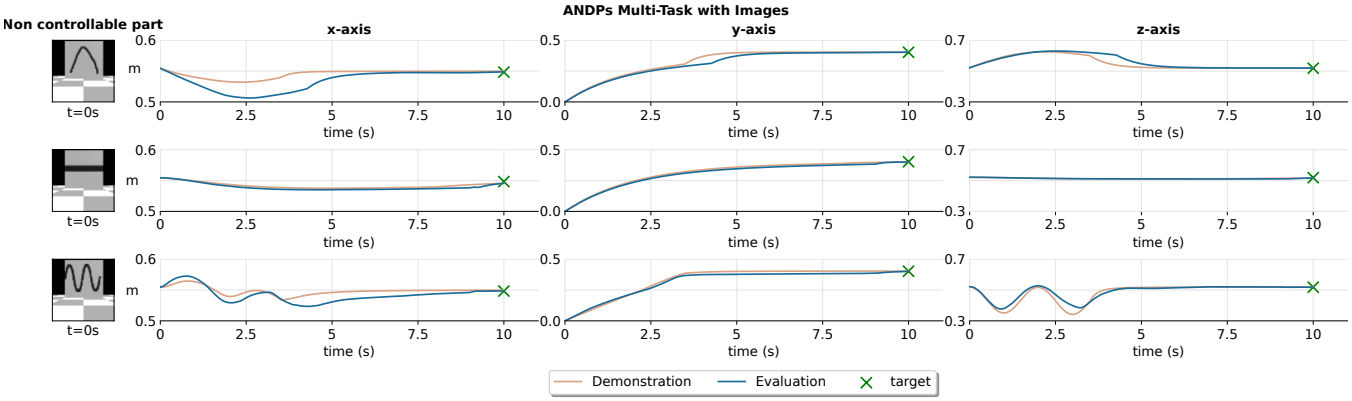

Fig. 4. Multi-task scenario with image inputs. All tasks are learned with a single model that can distinguish between tasks given an image input.

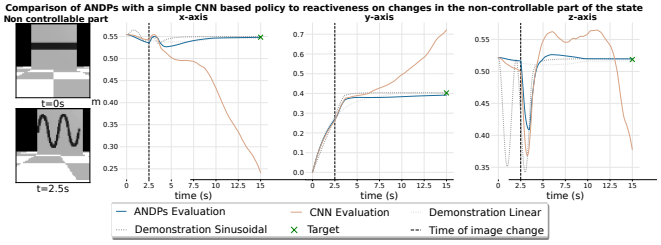

Fig. 5. Comparison of ANDPS with a simple CNN-based policy to reactiveness of changes in the non-controllable part of the state, we switch from the line image to the sine image at t=2.5 s.

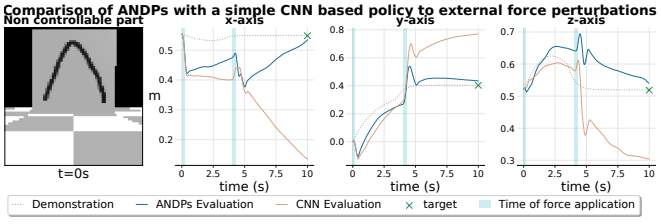

Fig. 6. Comparison of the reactiveness of ANDPs and simple CNNs to external force perturbations. We apply an external force twice: at $t = 0\,s$ and $t = 4\,s$.

2) Follow the Line: We perform a second experiment with a different robot platform to show the versatility of our proposed method. In this second experiment, we employ a quadrupedal robot for a visual path-following task: a GO1

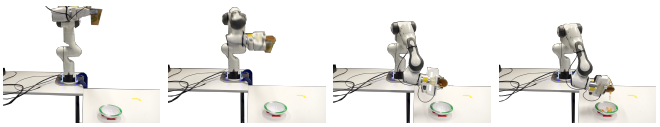

Fig. 7. From left to right, screenshots of a successful trial of the pouring task in the physical setting.

quadruped is required to follow an L-shaped path represented by the red and blue tape as shown in Fig. 8. To this aim, **we collected** 8 **demonstrations by teleoperating the robot using a Remote Control (RC) controller**.

We develop a motion policy based on ANDPs, where the state is defined by $\boldsymbol{x} = \{\boldsymbol{x}_{nc}, \boldsymbol{x}_c\} = \{\mathcal{I}, x, y, \theta\}$. Here $\boldsymbol{x}_{nc} = \mathcal{I} \in \mathbb{R}^{85\times85\times3}$ represents a scaled-down RGB image obtained from the GO1's front-facing camera and $\boldsymbol{x}_c = \{x, y, \theta\}$ denotes the plane's Cartesian coordinates and the robot's yaw orientation, which are captured using Phasespace, an active motion capture system. To evaluate the effectiveness of the motion policy, we conducted 18 experiments from various starting points. The outcomes, as depicted in Fig. 9, show that the trajectories generated by the learned model effectively reach their intended destinations, closely mirroring the patterns observed during training. The small regulation errors that can be observed in Fig. 9 are due to measurement noise and the GO1 high-level controller dead

zone which usually triggers an early stop when the control action is getting close to to zero.

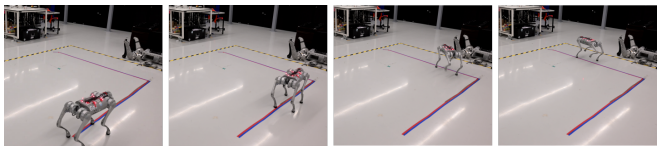

Fig. 8. From left to right, screenshots of the quadruped performing the line following experiment.

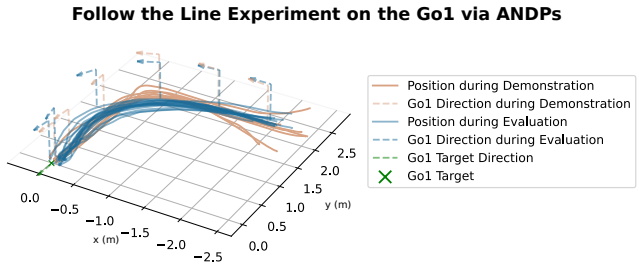

Fig. 9. A 3-dimensional display of the x, y trajectory collected during the training (in orange) and the evaluation (in blue) of the ANDPs policy. the arrows on the top part of the picture represent the robot orientation in correspondence with specific x, y locations.

VI. Conclusions

ANDPs are one of the first policy structures for robot learning that are general purpose while ensuring asymptotic stability of the produced behaviors. Using ANDPs, we successfully learned diverse tasks with various action space parameterizations and input types. While our experiments focused on IL/LfD scenarios, ANDPs are fully differentiable and versatile, making them applicable to pure RL settings, which we plan to explore in future work.

Another important feature of ANDPs is their inherent explainability. Since the underlying policy is a sum of elementary linear DSs, one can examine the ANDPs via analytical tools for understanding the reasoning behind the policy's decisions. We aim to investigate this in future work.

The main limitation of ANDPs is the reliance on a fixed attractor, which remains static throughout the episode. This poses challenges for learning long-horizon tasks and requires more complex optimization to relax constraints and allow non-monotonic motions. In future work, we plan to explore moving attractors or using state-of-the-art mappings that offer a promising pathway to address this limitation, enhancing flexibility in motion representation.

Acknowledgments

K. Chatzilygeroudis was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "3rd Call for H.F.R.I. Research Projects to support Post-Doctoral Researchers" (Project Acronym: NOSALRO, Project Number: 7541). D. Kanoulas and V. Modugno were supported by the UKRI Future Leaders Fellowship [MR/V025333/1] (RoboHike). For the purpose of Open Access, the authors have applied a CC BY public copyright license to any Author Accepted Manuscript version arising from this submission.

References

[1] P. Varin, L. Grossman, and S. Kuindersma, "A comparison of action spaces for learning manipulation tasks," in *IROS*, 2019.

[2] K. Chatzilygeroudis *et al.*, "A survey on policy search algorithms for learning robot controllers in a handful of trials," *IEEE Transactions on Robotics*, 2019.

[3] R. Martín-Martín *et al.*, "Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks," in *IROS*, 2019.

[4] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[5] A. Rajeswaran *et al.*, "Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations," in *RSS*, 2018.

[6] A. Billard *et al.*, "Robot programming by demonstration," in *Springer handbook of robotics*. Springer, 2008, pp. 1371–1394.

[7] S. M. Khansari-Zadeh and A. Billard, "Learning stable nonlinear dynamical systems with gaussian mixture models," *IEEE Transactions on Robotics*, 2011.

[8] F. Stulp and O. Sigaud, "Robot skill learning: From reinforcement learning to evolution strategies," *Paladyn, Journal of Behavioral Robotics*, 2013.

[9] S. Schaal, "Dynamic movement primitives-a framework for motor control in humans and humanoid robotics," in *Adaptive motion of animals and machines*, 2006.

[10] A. J. Ijspeert *et al.*, "Dynamical movement primitives: learning attractor models for motor behaviors," *Neural computation*, 2013.

[11] F. Khadivar, I. Lauzana, and A. Billard, "Learning dynamical systems with bifurcations," *Robotics and Autonomous Systems*, 2021.

[12] A. Ude *et al.*, "Task-specific generalization of discrete and periodic dynamic movement primitives," *IEEE Transactions on Robotics*, 2010.

[13] A. Ude, B. Nemec, J. Morimoto, *et al.*, "Trajectory representation by nonlinear scaling of dynamic movement primitives," in *IROS*, 2016.

[14] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," in *NIPS*, 2003.

[15] N. Figueroa and A. Billard, "A physically-consistent bayesian non-parametric mixture model for dynamical system learning." in *CoRL*, 2018.

[16] W. Amanhoud, M. Khoramshahi, and A. Billard, "A dynamical system approach to motion and force generation in contact tasks." RSS, 2019.

[17] S. Bahl, M. Mukadam, *et al.*, "Neural dynamic policies for end-to-end sensorimotor learning," in *NeurIPS*, 2020.

[18] S. Bahl, A. Gupta, and D. Pathak, "Hierarchical neural dynamic policies," in *RSS*, 2021.

[19] P. Fidelman and P. Stone, "Learning ball acquisition on a physical robot," in *ISRA*, 2004.

[20] F. Stulp, G. Raiola, *et al.*, "Learning Compact Parameterized Skills with a Single Regression," in *Humanoids*, 2013.

[21] F. Guenter *et al.*, "Reinforcement learning for imitating constrained reaching movements," *Advanced Robotics*, 2007.

[22] Y. Shavit *et al.*, "Learning augmented joint-space task-oriented dynamical systems: a linear parameter varying and synergetic control approach," *IEEE Robotics and Automation Letters*, 2018.

[23] N. Figueroa *et al.*, "A dynamical system approach for adaptive grasping, navigation and co-manipulation with humanoid robots," in *ICRA*, 2020.

[24] T. Osa *et al.*, "An algorithmic perspective on imitation learning," *Foundations and Trends® in Robotics*, 2018.

[25] E. D. Sontag, *Mathematical control theory: deterministic finite dimensional systems*. Springer Science & Business Media, 2013, vol. 6.

[26] A. Isidori, *Nonlinear control systems: an introduction*. Springer, 1985.

[27] Z. Artstein, "Stabilization with relaxed controls," *Nonlinear Analysis: Theory, Methods & Applications*, vol. 7, no. 11, pp. 1163–1173, 1983.

[28] Z. Jiang, Y. Lin, and Y. Wang, "Stabilization of nonlinear time-varying systems: a control lyapunov function approach," *Journal of Systems Science and Complexity*, vol. 22, no. 4, pp. 683–696, 2009.

[29] N. D. Ratliff, J. Issac, D. Kappler, S. Birchfield, and D. Fox, "Riemannian motion policies," *arXiv preprint arXiv:1801.02854*, 2018.

[30] B. Fichera and A. Billard, "Linearization and identification of multiple-attractor dynamical systems through laplacian eigenmaps," *Journal of Machine Learning Research*, vol. 23, no. 294, pp. 1–35, 2022.

[31] K. Chatzilygeroudis, D. Totsila, and J.-B. Mouret, "RobotDART: a versatile robot simulator for robotics and machine learning researchers," *Journal of Open Source Software*, vol. 9, no. 102, p. 6771, 2024.

[32] J. Lee *et al.*, "Dart: Dynamic animation and robotics toolkit," *Journal of Open Source Software*, 2018.