DIAG

Learning Multistable Motion Primitives from Demonstration via Multi-Attractor Dynamics

Academic Year 2024/2025

Professor:
Roberto Capobianco

External Advisor:
Cosimo Della Santina

Student:
Gabriele Giuseppe Di Marzo

n° 2012633

SAPIENZA
UNIVERSITÀ DI ROMA

SAPIENZA
UNIVERSITÀ DI ROMA

DEPARTMENT OF COMPUTER, CONTROL AND MANAGEMENT
ENGINEERING ANTONIO RUBERTI

Learning Multistable Motion Primitives from Demonstration via Multi-Attractor Dynamics

Professor:
Roberto Capobianco
External Advisor:
Cosimo Della Santina

Student:
Gabriele Giuseppe
Di Marzo

Academic Year 2024/2025

Contents

Learning Multistable Motion Primitives from Demonstration via Multi-Attractor Dynamics

Abstract

In Imitation Learning, generating velocity or acceleration commands from human demonstration trajectories is a crucial yet challenging task. While neural models have the potential to generalize across multiple demonstrations and target points, they often lack inherent stability—a fundamental requirement for industrial applications. Existing stability enforcement methods typically restrict the ability to learn from diverse demonstrations. To address this challenge, we propose a novel approach to Dynamic Movement Primitives (DMP) that integrates multi-stable dynamics, enabling both flexible and stable robotic motion learning. Our framework leverages neural models grounded in diffeomorphism learning to accommodate both discrete goal points and continuous curves of attraction, acknowledging their distinct dynamic characteristics. For discrete tasks, we develop a latent-space model with multi-stable dynamics, ensuring convergence to multiple predefined points. For continuous tasks, we extend the notion of discrete attractors to continuous attraction curves, enabling entire contours of target objects to function as stable attractors—broadening the scope of grasping points for robotic manipulators. We rigorously ensure stability through tailored loss functions and contrastive learning, fostering robust and reliable trajectory imitation. Experiments with a KUKA iiwa7 manipulator equipped with an artificial hand demonstrate the effectiveness of our approach, replicating human-demonstrated trajectories with stability under varied conditions.

1 Introduction

Learning by Demonstration (LfD) is a well-established approach for teaching specific skills to robots. Rooted in classical machine learning, it frames the problem from the perspective of imitation, where the learning system attempts to replicate the demonstrations provided by an expert. Unlike Reinforcement Learning, which requires an agent to extract behavior through trial-and-error interactions with its environment, LfD introduces an explicit bias toward optimal execution by leveraging expert demonstrations—placing it within the broader family of *Supervised Learning*. This framework offers practical solutions for integrating autonomous systems into industrial environments by enabling rapid training and efficient deployment. In a typical LfD setup, a human operator demonstrates the desired movement, and the robot learns from this input. When the operator physically guides the robot (through physical contact with it) through the motion — recording its internal states during the process — the method is referred to as kinesthetic teaching, which is generally feasible under an impedance control scheme.

Recent developments in Learning from Demonstration (LfD) have explored the use of visual information, where an operator equipped with a camera captures the simulated point of view of the robot—typically a manipulator—as it will experience it during task execution. These approaches commonly rely on large neural network models to interpret and learn directly from image data. While this vision-based paradigm enables more flexible and intuitive demonstrations, it comes with significant challenges. Chief among them is the vast amount of data required to extract meaningful patterns from visual scenes, which often makes training computationally expensive and time-consuming. Furthermore, such models generally offer no guarantees regarding the reliability or consistency of visual inputs during future executions, limiting their robustness in dynamic or unpredictable environments. We report that the field of LfD is converging toward a mixed approach, where different modality of teaching are combined with the aims to collect different data source[1].

Another line of research addresses the concept of rewards by training an external system to infer a reward function based on the distance from a reference trajectory. This approach falls within the framework of Inverse Reinforcement Learning (IRL) [2]. As with most Reinforcement Learning (RL) methodologies, however, IRL presents significant challenges when applied in real-world scenarios. The need for extensive interaction with the environment to train the models makes the process complex and resource-intensive. Training in simulation often results in a model that does not accurately reflect real-world dynamics, while direct interaction with physical robots can be dangerous, time-consuming, and operationally demanding.

Furthermore, when the learning objective is to directly predict the low-level inputs of the robot, the resulting policies tend to be task-specific and difficult to transfer across different platforms or tasks. To address this limitation, many recent approaches adopt a modular structure, where a learned model predicts high-level desired states or trajectories, and a separate low-level controller is responsible for converting these predictions into executable motor commands. This separation promotes reusability, and enhances generalization across the different robots.

A substantial body of literature approaches the problem from a control-theoretic perspective, aiming to design learning systems that exhibit stability properties akin to those found in model-based control techniques. This includes methodologies based on Lyapunov theory, contraction analysis, and diffeomorphism-based approaches.
Viewed through this lens, one of the primary limitations lies in the difficulty of generalizing these methods to accommodate multiple and diverse demonstrations while maintaining consistent stability properties. Many methods in this category are also difficult to scale to complex or unconstrained architectures, as they often require restrictive assumptions on the learning model—such as the positive semi-definiteness of certain functions or the invertibility of the learned mappings.

In this thesis, we focus on designing a learning framework that preserves stability across different demonstrations. To achieve this, we employ a diffeomorphic mapping defined over a multi-stable dynamical system, represented in the latent space of a neural network. This approach serves as the foundation for a Motion Primitive (MP) framework. The concept of motion primitives, originally introduced by Jensen [1], views behavioral skills of an agent as stable dynamic systems.
By leveraging multiple latent dynamics and encoding appropriate biases through the design of the loss function, this work contributes the following:

- We explore strategies for **integrating neural models with dynamical systems**, aiming to preserve control-theoretic properties within learning-based frameworks.
- We present a **diffeomorphism-based neural Dynamic Movement Primitive (DMP) framework capable of modeling multiple discrete attractor dynamics**. This formulation allows the system to generalize across different demonstrated targets—such as varying grasping points—by embedding them as distinct stable attractors in the latent space.
- We extend this framework to represent **continuous closed-loop attractor curves**, generalizing the concept of discrete grasping points into continuous targets, represented by a contour in the end-effector space.

In the remaining part of this chapter, we will proceed to review the foundations of

the Dynamic Movement Primitive (DMP) methodology, the existing literature on the stability of neural models, and the theoretical background that underpins our research.
In Chapter 2 we illustrate the datasets that we collect for our experiment, alongside the preprocessing utilize during the simulation and the real world experiment.
Chapter 3 present the main contribute of the thesis, and develop 3 different solution for solve the multi-attractor problem. After having evaluated the startegy used in the comparison in Chapter 4, we jump into the results of the thesis, in Chapter 5. There, the different experiments and the ablation study will guide toward the final implementation of the framework. Moreover, we illustrate in the same chapter the real-world setting and the robotics experiment.
In Chapter 6, we highlight the limits of the proposed appraoch and indicate some possible future upgrade. Finally, Chapter 7 will ends this research by statement the goals reached during the research.

1.1 Background

1.1.1 DMP

Dynamic Movement Primitives (DMPs) are a set of methods for trajectory control and planning. The first formulation dates back to 2002, in the work of Schaal et al. [3]. They were initially used in the field of humanoid robotics, where the goal was to mimic specific motion patterns. Subsequently, DMPs were further developed through the works of Ijspeert, starting from his seminal paper [4]. In this work, the problem of learning from human demonstrations is addressed to learn motor control while mapping complex behaviors to simpler dynamic systems. Several objectives were considered when defining his solution; in particular, Ijspeert outlined seven requirements that his system should satisfy. These requirements are as follows:

1 The system should be able to learn point attractor along with the limit cycle attractors.

2 The model should be time-indipendent

3 the model should be able to handle multi-dimensional dynamical systems

4 The tuning of the parameters should be as simple as possible

5 The systems should incorporate coupling terms, as used in synchronization studies

6 The system should allow real-time computation

7 Scale and temporal invariance is a desirable requirements, and the change in the task space should not modify the attractor landscape

Considering these points, the proposed formulation for the dynamics was:

$$\tau\ddot{y} = \alpha_z(\beta_z(g - y) - \dot{y}) + f \tag{1}$$

where f is define as the summation of N $\Psi(t)$ function, where each $\Psi(t)$ represents an exponential basis function:

$$f(t) = \frac{\sum_{i=0}^{N} \Psi_i(t) w_i}{\sum_{i=0}^{N} \Psi_i(t)}$$

$$\Psi_i(t) = \exp\left(-\frac{1}{2\sigma_i^2}(x - c_i)^2\right)$$

The terms σ and c are constants to be determined, representing the width and the center of the basis function, respectively.
This formulation allows the system to describe a point attractor, with the system's equilibrium at $y(t) = g$. The exponential basis is used to learn the correct trajectory execution from demonstrations.
To incorporate rhythmic behavior, such as limit cycles, the exponential function is modified as follows:

$$\Psi(t) = \exp(h(\cos(\phi - c) - 1))$$

The forcing term is then modified accordingly, using r, the amplitude signal, instead of the equilibrium term $(g - y)$. Over time, the forcing term will vanish, leading to a stable dynamic.

An interesting solution that falls under the concept of dynamic system learning and is capable of addressing multiple attractors is presented in [5]. In this work, the space of a multi-stable dynamical system is composed of linear subsystems, each assigned to a specific region of the workspace through a *support vector machine* (SVM) classifier. Moreover, this paper offers a compelling perspective that can be enhanced by neural approaches, as the SVM-based method suffers from discontinuities between different regions of the space.
Several works have extended the seminal work of Jispeer by introducing neural approximators to learn the forcing terms or even the entire dynamics. We first report the work of Si, Wang, and Yang in [6], where a neural model is used to approximate the response to a payload change. The demonstrations were collected using a remote device that enable teleoperation.
Moreover, different input modality can be taken into account by using neural model, like in [[7], [8]], where CNN architecture are used for address input or feature extraction. Usually, before the integration of CNN architecture, a double stage procedure was required, where the input feature was first extracted from the image with standard feature extraction tool, and then maps into the DMP parameters. Modern work bypass

this double stage setting by optimizing directly the learning function for extract the parameters by himself, enable a End-to-End learning [8]
However, integrate neural function require special solution for impose stability condition, usually not matched by standard neural modules like MLP or CNN. Several approaches have addressed this limitation by adapting the learner's structure to ensure compatibility with stability requirements. To explore this potential, the DMP space is frequently mapped to the hidden state of a neural model, where stability properties can be enforced on the latent state variables.
The first work that introduce a different space for the DMP parameters was [9], which addressed DMPs by projecting task space variables into a latent state obtained from a Gaussian Process Latent Variable Model (GPLVM). The system was then tuned within the latent space before being projected back to the task space. This approach enhances system robustness by imposing a regular structure in the latent space and incorporating dimensionality reduction, which aids in stability analysis. However, the paper emphasizes the importance of demonstrations, noting that the learned space is often not sufficiently smooth to enable translation to new tasks. This limitation arises from the function approximator itself, which assumes a Gaussian structure. Additional details on the stability technique for neural networks are given in the next section.
Neural models have addressed this issue by providing smoother and more nonlinear mappings. In particular, autoencoders [10] have been employed to establish relationships in cases where data do not follow a Gaussian distribution. To ensure that hidden units are activated only for specific inputs, sparsity is enforced via an l_1-norm penalty on the loss term. The authors claim that this approach allows for the interpolation of demonstrations to generate novel motions.

1.1.2 Stability for neural models

As discussed in the previous section, stability is a cornerstone in robotic systems, particularly when interacting with humans. In the literature, there are three main approaches to enforcing stability conditions [11]:

- Lyapunov-based methods [[8], [12], [13], [14]]
- Contraction theory [[15], [16]]
- Diffeomorphism mapping [[17], [18], [19], [20]]

Starting with the Lyapunov-based methodology, a classical approach in the field is SEDS [13], which ensures global asymptotic stability by modeling the joint distribution of positions and velocities using a Gaussian Mixture Model (GMM) and extracting a smooth velocity field through Gaussian Mixture Regression (GMR). To guarantee that all trajectories converge to a predefined attractor, SEDS incorporates *stability constraints inspired by Lyapunov theory* during GMM parameter estimation. Specifically,

it enforces that the velocity field satisfies the Lyapunov condition by ensuring the inner product $(\mathbf{x} - \mathbf{x}_g)^\top f(\mathbf{x}) < 0$ for all $\mathbf{x} \neq \mathbf{x}_g$, corresponding to a negative time derivative of the Lyapunov function $V(\mathbf{x}) = \frac{1}{2}\|\mathbf{x} - \mathbf{x}_g\|^2$. This guarantees that the system's energy decreases over time, leading to global asymptotic stability of the attractor and preventing undesirable behaviors such as oscillations or divergence.
A more recent Lyapunov-based approach for stability in neural dynamical systems is presented by Sochopoulos et al. [8] with their method called StableNODE. To the best of our knowledge, this is the only neural architecture capable of handling multi-attractor systems. The approach defines a nominal dynamics function $f_\theta(x)$ along with a corrective term $u(x)$, designed to ensure stability in the Lyapunov sense. The modified dynamics $f(x) = f_\theta(x) + u(x)$ preserve the original dynamics when the Lyapunov condition $L(x) \leq 0$ is satisfied and introduce corrective control otherwise to enforce exponential stability. All components, including the Lyapunov function and encoder-decoder mappings, are jointly optimized using a Neural ODE loss. Further details on this method are provided in the evaluation strategy section 4.

In the context of *contraction theory*, recent advancements have utilized contraction theory to enhance stability and robustness in imitation learning and dynamical systems modeling. Contraction theory offers a differential framework for ensuring *incremental stability*, guaranteeing that all trajectories converge exponentially toward each other, regardless of initial conditions. In imitation learning, [15] incorporated contraction conditions into policy learning, enabling robust and safe *online motion planning* with formal guarantees of trajectory convergence and safety filtering.
Despite its advantages, contraction-based methods often impose structural requirements on the learned function. In particular, the dynamics or policy must be differentiable to allow Jacobian computation, and in some instances, must adopt specific forms such as control-affine systems. Additionally, contraction theory often involves a Riemannian metric $M(x) \succ 0$, which may either be fixed or parameterized as part of the learning process. This requires joint optimization over both the dynamics and the metric, motivating the adoption of structured neural architectures that facilitate contraction analysis, such as networks with residual connections, monotonicity properties, or explicit stability-promoting layers.

The third class of methodologies for enforcing stability is rooted in diffeomorphism mapping. A diffeomorphism is an invertible mapping between two spaces, associating each point in the source space with a unique equivalent point in the target space. These methods inherently require structural priors since most approaches depend on the invertibility of the learned function [19].
In the following section, we introduce our reference architecture, which leverages diffeomorphic stability conditions. Therefore, the specific details of this methodology

are discussed in the next paragraph.

1.1.3 Reference Architectures

The two architectures that serve as the foundation for our experiments are **Condor** [17] and **Puma** [18], both developed by Perez-Dattari in collaboration with TU Delft. These models utilize a diffeomorphic mapping to approximate single-attractor systems. While Condor explicitly models latent dynamics through the equation $\dot{y}(t) = \alpha(y_G - y(t))$, Puma instead derives the latent state evolution by using the gradient of the neural encoder weights as dynamical system.
Both architectures are grounded in contrastive learning, and their respective stability conditions are detailed in the original publications.
From an architectural standpoint, Condor and Puma share a similar structure, comprising an encoder and a decoder, both implemented using simple multi-layer perceptrons (MLPs). Crucially, the methodology adopted by both models is fully general and does not rely on specific constraints imposed on the function approximators. This design choice enhances the flexibility of the framework and facilitates potential future integration with more complex components, such as transformers or convolutional neural networks.
The encoder block $\phi(x)$ and the decoder block $\psi(y)$ are each composed of three layers with 100 neurons per layer, resulting in a total of 600 neurons per network. Computational efficiency is a key consideration, enabling these models to be trained and executed within a short time frame. As illustrated in Figure 1, the input to both models is the end-effector state $x(t)$, and the output corresponds to its time derivative $\dot{x}(t)$. The latent state $y(t)$ can be obtained either by projecting the input state through the encoder network $\phi(x(t))$, or by evolving an initial latent state according to the predefined dynamics $\dot{y} = g(y)$, where $g(y)$ represent the hard-coded system of Condor or the gradient of the encoder in Puma.

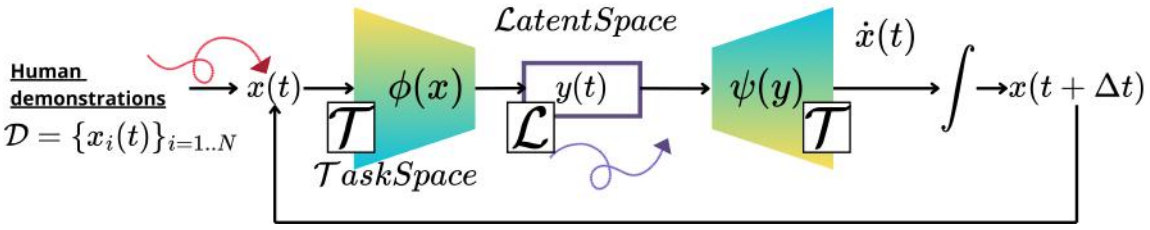


Figure 1: Basic scheme of the Condor and Puma Architecture

- the learned evolution in the task space, $f_\theta^{\mathcal{T}}(x) = x(t)$
- the projected evolution in the latent space, $f_\theta^{\mathcal{T}\to\mathcal{L}}(x) = y(t)$
- the analytically defined dynamics of the latent state, $f^{\mathcal{L}}(y) = y(t)$

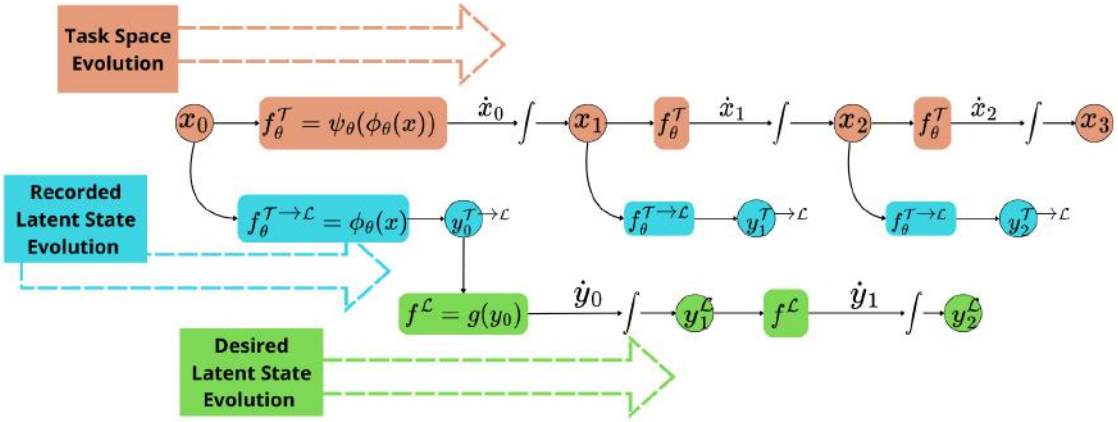


Figure 2: Trajectory evolution of the 3 systems used in Condor and Puma over time

Stability in both architectures is enforced through a dedicated **stability loss**. In the case of Condor, this takes the form of a contrastive loss, whereas in Puma, stability is enforced using a variant of the triplet loss.

A standard formulation of contrastive loss is often introduced in the context of a binary classification task. Consider a generic data point z_i and its corresponding class label c_i. The contrastive loss aims to minimize the distance between points of the same class, while maximizing the distance between points of different classes. This concept was initially applied in unsupervised learning, where the objective is to organize an embedding space such that semantically similar samples are grouped together, and dissimilar samples are well-separated. By "organize," we refer to partitioning the space of dissimilar concepts while preserving a meaningful internal structure.

The contrastive loss typically operates on an anchor point (serving as a reference) and a set of comparison points, categorized as positive samples (similar to the anchor) and negative samples (dissimilar). The loss function is defined as follows:

$$\mathcal{L}(z_i, z_j, \theta) = \mathbb{1}[c_i = c_j]|f_\theta(z_i) - f_\theta(z_j)|_2^2 + \mathbb{1}[c_i \neq c_j] \max(0, \epsilon - |f\theta(z_i) - f_\theta(z_j)|_2)^2 \quad (2)$$

Here, z_i and z_j represent input samples, while c_i and c_j are their corresponding class labels. The indicator functions $\mathbb{1}[\cdot]$ determine whether the samples belong to the same class or not. The margin ϵ acts as a tolerance parameter, ensuring that negative pairs already sufficiently separated in the embedding space are not further penalized.

Condor leverages this formulation to model its stability property through a contrastive loss framework.

For having a proper diffeomorphism, two condition must be respected:

- $f_\theta^{\mathcal{T}\to\mathcal{L}}(x(t)) = f^{\mathcal{L}}(y^{\mathcal{L}}(t))$
- $\psi_\theta(x(t)) = y_g \implies x_t = x_g$

The first condition is used to ensure that the evolution of the latent dynamic is consistent with the evolution of the task dynamic. The second one ensure that only the task-attractor is mapped to the latent-attractor. The **first condition** is addressed by minimizing the distance between the evolution of the task space and the corresponding evolution in the latent space. This leads to the first component of the contrastive loss, defined as:

$$l_{\text{match}} = d(y_t^{\mathcal{L}}, y_t^{\mathcal{T}})$$

where $d(\cdot,\cdot)$ is a distance function. As is common in the literature, the L_2 **norm** is selected as the distance metric.
The **second condition** is more challenging to enforce directly. To address this, a **surrogate stability condition** is introduced:

$$y_{t-1}^{\mathcal{T}} \neq y_t^{\mathcal{T}}, \quad \forall x_t \in \mathcal{T} \setminus \{x_g\}$$

This condition ensures that the latent representation changes over time, except at the goal state x_g. Based on this surrogate condition, the second term of the contrastive loss is defined as:

$$l_{\text{sep}} = \max(0, m - d(y_{t-1}^{\mathcal{T}}, y_t^{\mathcal{T}}))^2, \quad \forall y_{t-1}^{\mathcal{T}}, y_t^{\mathcal{T}} \in \mathcal{L}$$

We notice that this term don't model exactly the case in which we reach the goals state, while the authors report that the sampling of that point is highly unlikely.
Having define the 2 component, we can define the stability loss of Condor:

$$\mathcal{L}_{stable} = \sum_{b=0}^{B-1} \sum_{t=1}^{H} ||y_{t,b}^{\mathcal{L}} - y_{t,b}^{\mathcal{T}}||_2^2 + max(0, m - ||y_{t-1,b}^{\mathcal{T}} - y_{t,b}^{\mathcal{T}}||_2)^2 \tag{3}$$

Puma adopts a different approach by relaxing the stability condition in favor of enforcing the diffeomorphism property. By reformulating the same starting condition as Condor, it can be demonstrated that for any $y_t \neq y_g$, the distance $d_t(y_g, y_t)$ must decrease over time, resulting in $d_t > d_{t+\Delta t}$. This distance should remain constant once the equilibrium condition is reached. Based on this reasoning, the stability loss for Puma is reformulated as:

$$\mathcal{L}_{\text{stable}} = \sum_{b=0}^{B-1} \sum_{t=1}^{H} \max[0, \delta(y_{g_i}, y(t+\Delta t)) - \delta(y_{g_i}, y(t))]$$

where $\delta(\cdot,\cdot)$ represents the distance metric. A different symbol is used for the distance function because Puma extends the previous implementation by introducing distinct

metrics for different spaces. This concept of distance functions leads to a new topology, which defines a manifold space in which the trajectories are described. In particular, the Euclidean distance is replaced with spherical distance.
It is also important to note that the mapping of the latent state derivative, $f^{\mathcal{L}}$, is obtained by the gradient of the encoder, as shown below:

$$\dot{y}(t) = \frac{\partial \psi_\theta(x(t))}{\partial t}$$

The key advantage of Puma and Condor, compared to existing solutions in the literature, is that neither method imposes structural constraints on the approximation function. This flexibility results in a **fully generalizable framework** capable of extending to any architecture. Moreover, the dimensionality of the latent state and the task state need not be the same, enabling exploration within the latent space.
Finally, we consider the **imitation loss**, denoted as $\mathcal{L}_{\text{imit}} = \text{MSE}(\hat{x}, x_{\text{demo}})$, where $\hat{x}$ represents the trajectory generated from the same starting point as the demonstrations, and x_{demo} refers to the actual demonstration trajectory.
The overall loss for both systems can be defined as:

$$\mathcal{L} = \mathcal{L}_{\text{imit}} + \mathcal{L}_{\text{stab}}$$

where $\mathcal{L}_{\text{stab}}$ represents the stability loss, which has distinct formulations for each system.
A limitation of this methodology is its reliance on a single attractor dynamic, which this thesis aims to address.

1.2 Research Statement

The primary objective of this research is to integrate multi-equilibria dynamics into the latent state of a neural model within the Dynamic Movement Primitives (DMP) framework. This approach addresses the challenge of learning multiple stable trajectories while maintaining flexibility and robustness in dynamic environments.
Learning multiple trajectories while ensuring stability is a significant challenge in robotics. To the best of our knowledge, few works have explored the feasibility of this integration, and since the beginning of our research, only one study [8] has successfully addressed the problem of multiple attraction points within the DMP framework.
Traditional methods often impose constraints on the model's structure, which limits their applicability to more complex, real-world scenarios. Furthermore, many neural solutions require extensive datasets, making them impractical for robotic systems that need to generalize from a limited number of demonstrations.
Our research focuses on developing strategies that enable learning multiple trajectories while ensuring the stability of the learned dynamics. We aim to maintain flexibility by not constraining the learner function to any specific structure, allowing us to

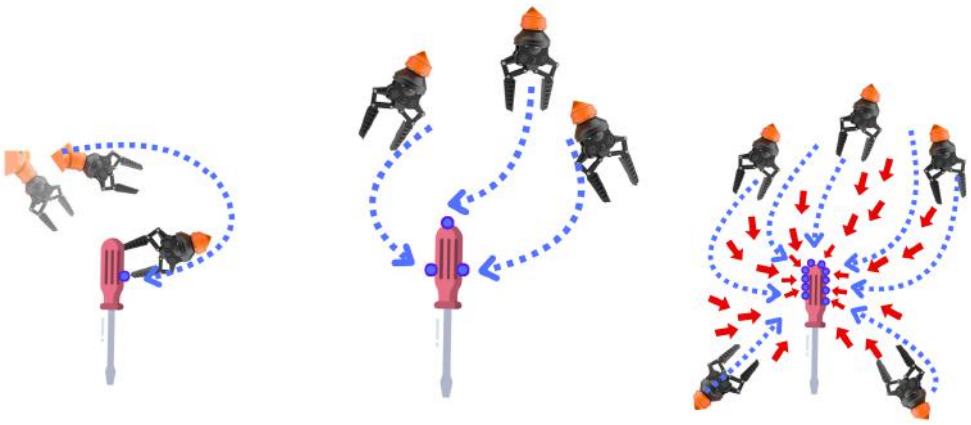


Figure 3: Comparison between the effect of single attractor and multiple attractors in a generic grasping task

explore various methodologies that can be seamlessly integrated with more complex architectures.
This integration will enable the system not only to learn multiple trajectories across different regions of the workspace but also to handle similar trajectories or attraction points, effectively generalizing the concept of grasping point
The ultimate goal is to enhance a robotic manipulator with the ability to autonomously select the most suitable goal from a set of possible options, treating each as equally important. To improve the interpretability of our results, we plan to incorporate prior knowledge from physics and system theory into our model. The expected result are visible in fig 3

2 Data Management

In this section, we present the strategy adopted for data collection, as well as the preprocessing steps applied in both the discrete and continuous settings.

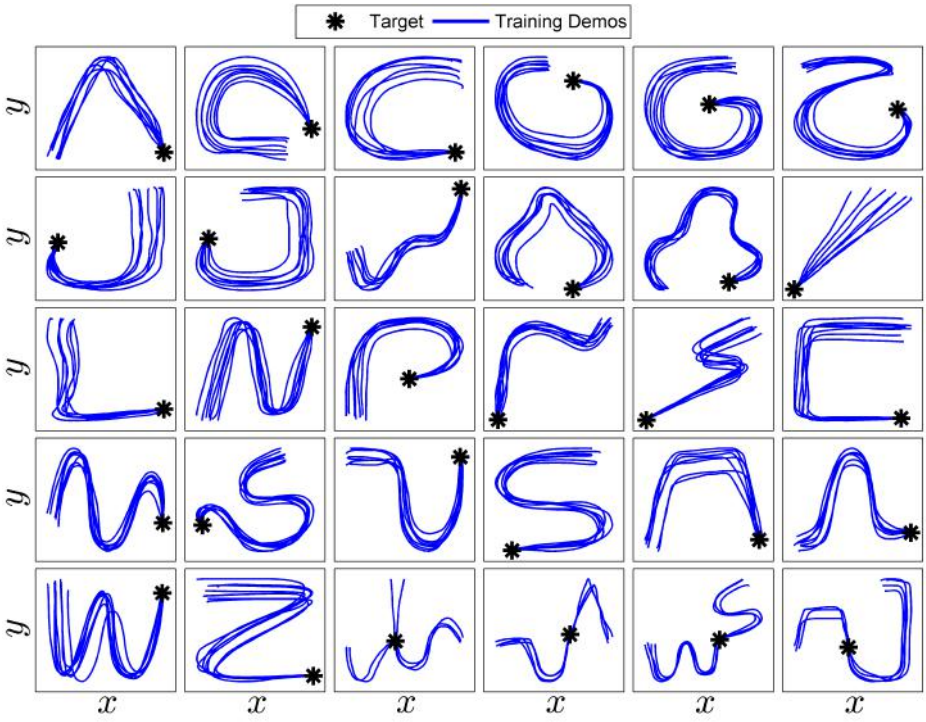


Figure 4: The LASA Handwriting dataset, described in [21]

2.1 Datasets and Trajectory Collection

The primary input to our model is the end-effector state. To this end, we utilize a collection of Cartesian trajectories. A widely adopted benchmark in DMP is the LASA dataset [21], which comprises 30 human handwriting motions recorded using a tablet. For each motion, the dataset provides 7 demonstrations of the desired trajectory, each beginning from a distinct initial position. The state is represented as 2D positional data. Examples from the LASA dataset are shown in Fig. 4.

Works focused on multi-stable learning [8] often employ variations of the LASA dataset, typically by stacking multiple demonstrations together. However, this approach has a notable limitation: it partitions the demonstration space by assigning each trajectory to a distinct, non-overlapping region.

In our experiments, we collect a new set of demonstrations using a custom computer interface that records mouse pointer movements. The output is a collection of 2D trajectories. Unlike the standard single-equilibrium cases, the dataset used in our experiments presents additional challenges.

First, some trajectories originate from positions that are close to other trajectories,

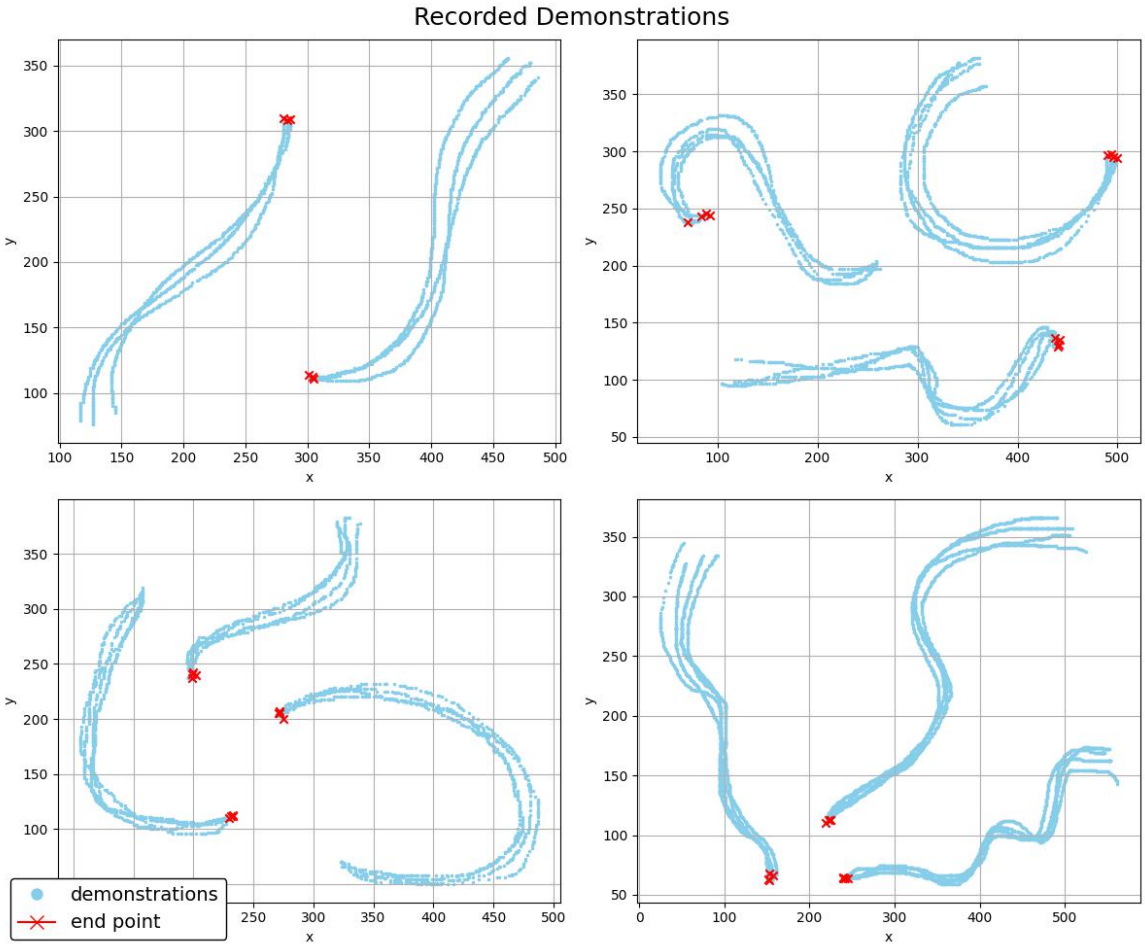


Figure 5: Example of the recorded demonstrations. Starting from the top left, we see a simple double attractors demonstrations set. On the top right, a case of 3 attractors with a trajectory that start near to the body of the others trajectories. In the bottom, we see 2 example of trajectory that converge toward the same area

leading to overlapping regions in the initial states. Second, in certain experiments, the goals are relatively close to each other, increasing the complexity of trajectory separation. Finally, some demonstrations introduce cyclic behaviors, altering the intra-trajectory distances before converging to the goal states. Examples of these behaviors are illustrated in Fig. 5.

2.1.1 Preprocessing

Once the demonstrations are collected, several features are extracted, including the goal positions and workspace boundaries. Given a fixed number of goals N, we use the `scikit-learn`[22] clustering routine [23] to compute the centroids of N clusters. These centroids are then used as the target goals.
Prior to training, the demonstrations are padded to enable the use of Backpropagation Through Time (BTT) [24]. Padding is performed by identifying the maximum trajectory length and appending zero vectors to the end of shorter trajectories. Subsequently, all trajectories are normalized sequentially to the range $[-1, +1]$, and subsampled to ensure a uniform distance between points. To mitigate the noise introduced by this subsampling process, the selected points are smoothed using a spline interpolation routine from SciPy [25].
Finally, the trajectories are replicated and segmented into imitation windows, enabling BTT to operate efficiently over the temporal structure of the data.

2.1.2 Real world trajectory collection

For the robotics experiments, demonstrations were collected from a physical robot using kinesthetic teaching [1]. Using the KUKA interface within the ROS framework, we recorded the joint values along each trajectory. The corresponding end-effector trajectories were then computed using forward kinematics.
A similar preprocessing pipeline, as described above, was applied to these robotic demonstrations. Moreover, in scenarios where learning the orientation is required, the end-effector state is augmented with orientation information. While the ROS interface provides the end-effector orientation as rotation matrices, we convert these to quaternion representations for consistency within our framework.

2.2 Shape Processing

The continuous attraction curve case require some extra information that are extracted from a separated module. In particular, given an image, the contour of the object need to be extracted, expanded and scaled down for generate the hard negative points, and finally parametrize by a value in $[+\pi, -\pi]$. This section describe the pre-processing done for obtain the required data

2.2.1 Shape Contour and Hard Negative

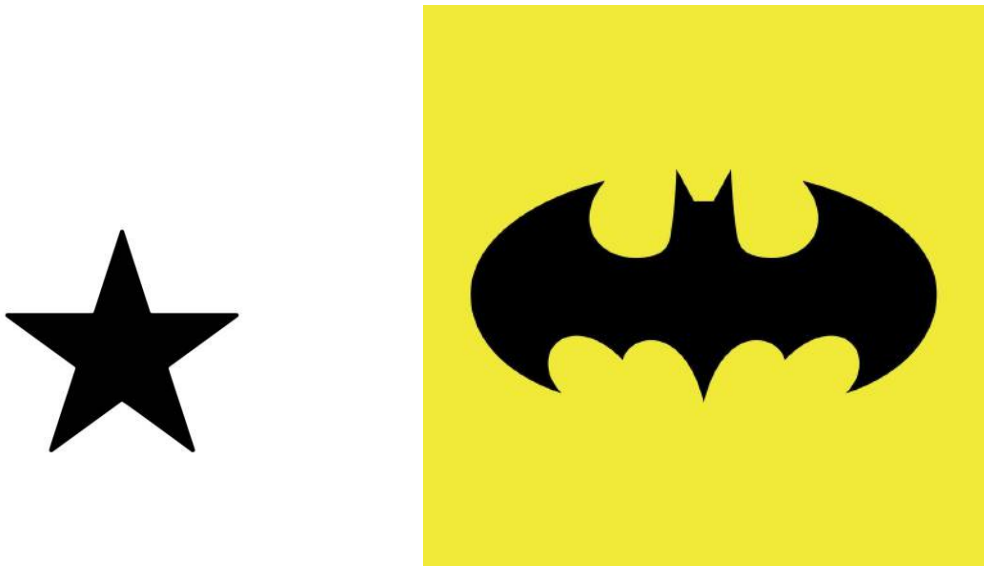


Figure 6: Example of the input images used in the Continuous Attractor case

This section describe the methodology used for the generation of the point that compose the continuous attraction curve describe in the relative section 3.2, and the hard negative generation strategy.
The first step, of course, is the extraction of the contour of the target shape. In complex environments, this can be solved using external segmentation methods, probably the best approach will be based on convolutional neural network. A standard pipeline for this could be divided in (1) object recognition, (2) segmentation and finally the (3) preprocessing. As this was out of the scope of this thesis, simpler image have been collected form the web (see Fig. 6) and used as input directly to the 3-th step.
Before describe the procedure for extract the relative data from the images (that represent our complex-shape case), we must report that our first solution has been the interpolation of the discrete attractor with simple geometric shapes (i.e. cirlce and triangle). In that case, the hard negative has been extracted by simpler scaling of the original object, while the point have been sample by using the analytical description of the shape

1. **Color Conversion**: The input RGB image is first converted to grayscale using OpenCV's `cv2.cvtColor()` [26] with the `COLOR_BGR2GRAY` flag, reducing the image to a single channel intensity representation.

2. **Binarization**: The grayscale image is thresholded using inverse binary thresholding (`THRESH_BINARY_INV + THRESH_OTSU`). This automatically determines the optimal threshold value to separate foreground from background while inverting the result (foreground becomes white).

3. **Contour Extraction**: The main object contour is identified by:
   - Finding all external contours using `cv2.findContours()` with `RETR_EXTERNAL` mode (retrieving only the outermost contours) and `CHAIN_APPROX_NONE` (storing all contour points)
   - Selecting the largest contour by area using `max(contours, key=cv2.contourArea)`
4. **Morphological Operations**: With this, we can extract the hard negative from our contour.
   - A 13×13 elliptical structuring element is created as the kernel for morphological transformations
   - **Dilation**: The binary image is progressively dilated in 3 iterations (using `cv2.dilate()`), each time extracting the outermost contour. This creates 3 concentric contours expanding outward from the main object.
   - **Erosion**: Similarly, the binary image is eroded in 1 iteration (using `cv2.erode()`) to create 1 contracted contour inside the main object.

Figure 7: Contour and Hard Negative extraction

The effect of this pipeline is visible in figure 7.

2.2.2 Orientation Mapping

As discussed in the final part of the continuous case, a second class of input data required by our models involves orientation mapping: a function that assigns each point along a shape's contour a corresponding location on the unit circle. This mapping encodes the orientation of the contour as it is traversed. To perform this operation, each point must be associated with a latent reference orientation, denoted by $\hat{\theta}$.
For simple or symmetric shapes, this can be achieved using a polar coordinate representation, where the shape is naturally parameterized by angular positions. However, for more complex geometries—such as the silhouette of the Pokémon figures —this method

becomes ineffective. Consider, for instance, the tail of such a figure: its intricate structure and asymmetry make it difficult to impose a consistent, meaningful angular ordering based on polar coordinates alone.

To address this, our preprocessing pipeline includes a step that reorganizes unordered contour points into a coherent, sequential representation. Given a set of N unordered 2D contour points, we apply a nearest-neighbor traversal algorithm to construct an ordered path through the shape. Starting from an initial seed point (typically the first in the input array), the algorithm iteratively selects the closest unvisited point, continuing until all points are incorporated. For efficiency, we utilize a k-dimensional tree (k-d tree) to enable fast spatial queries during the traversal.

Once the contour is ordered, we parametrize each point by an angular value $\hat{\theta}$, uniformly distributed over the interval $[-\pi, +\pi]$. This angular parametrization provides a smooth and continuous representation of the contour's traversal, enabling consistent orientation mapping (Fig 8). The sorted point and their orientation are saved and used as input data of the training procedure.

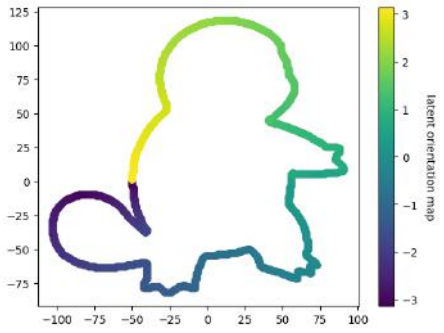


Figure 8: Associated orientation for each point of the contour

3 Methodology

Building on the Condor [17] and Puma [18] frameworks, we design a latent dynamic system capable of exhibiting multi-equilibrium behavior in a 300-dimensional space, corresponding to the hidden state dimensionality of the neural models. Our efforts focus on several cases: first, we explore the imitation of Cartesian reference trajectories, with equilibrium points located at the end of each demonstration. Following this principle, we construct continuous attraction curves using these equilibrium points. Finally, we develop a method based on the Puma architecture to handle multi-attractor dynamics. This last solution enables the model to learn not only position but also orientation in the task space, allowing full control over the robot's end-effector.

At inference time, the robot publishes the end-effector's position and orientation to a ROS topic. These values are converted into a suitable format and sent to the model, which then outputs reference velocities for an inverse kinematics control scheme. This controller closes the loop, enabling the robot to move continuously toward its goals.
Training details follow the integration pipeline described in Section 1.1.3. Both the latent dynamics $\dot{y}(t)$ and the task dynamics $\dot{x}(t)$ are integrated using a fixed-step Euler scheme over a defined time horizon. A comprehensive diagram of the architecture is shown in Fig. 9, while specialized variants for each case are provided in their respective sections.
The remainder of this chapter is organized as follows: the first part discusses the considerations and solutions for the discrete attractor case; the second part extends the framework to closed shapes; and finally, we address orientation control using the Puma model.
A brief note on a specific category of plots that will appear frequently in this section: as mentioned previously, the reference framework combines an imitation loss and a stability loss. The stability loss ensures that goals are reached independently of the trajectory. Several plots omit the imitation loss to evaluate the convergence properties of the framework quantitatively.

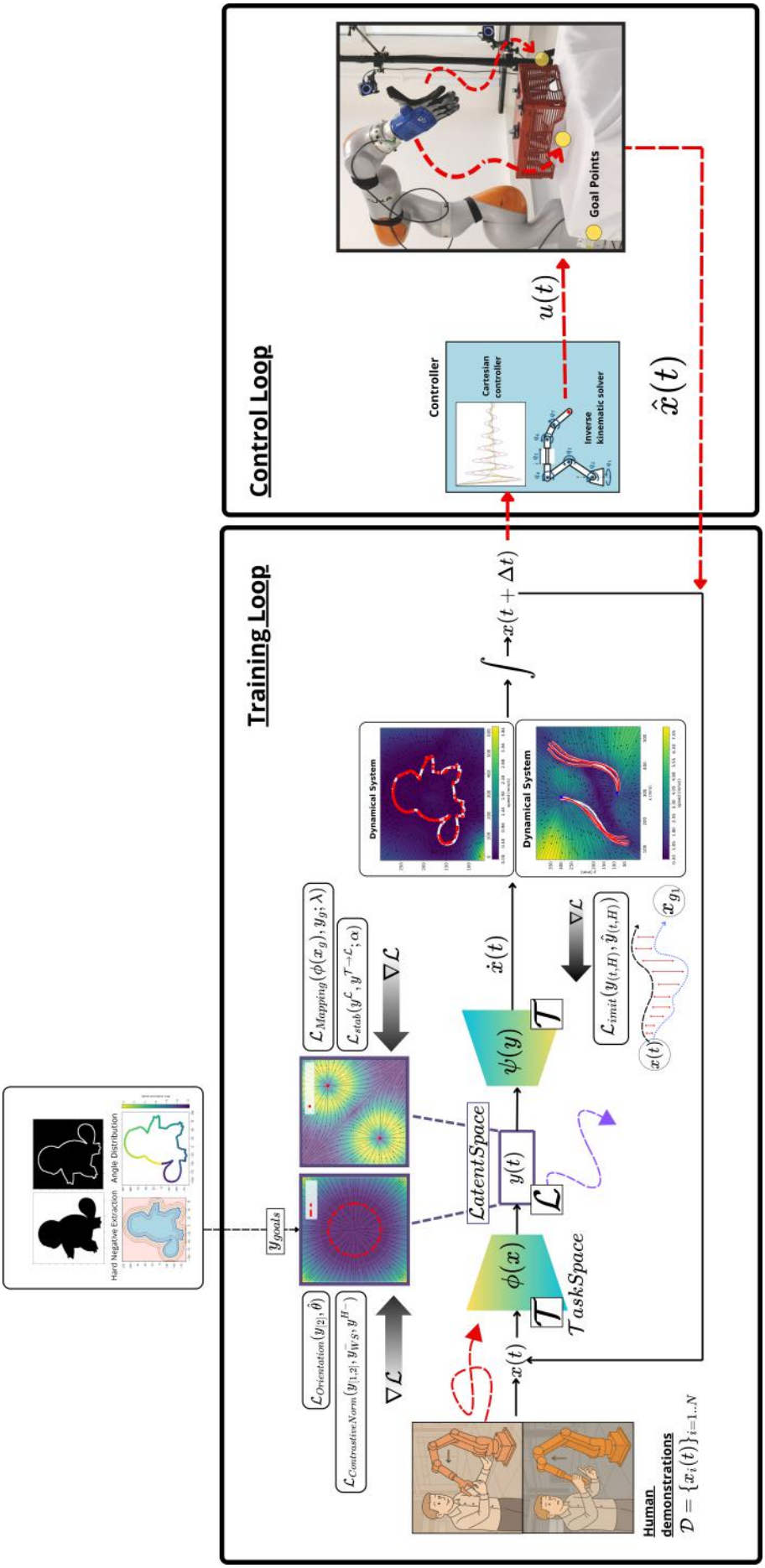


Figure 9: General architecture of proposed solution, with all the loss component and their point of application

3.1 Discrete Case

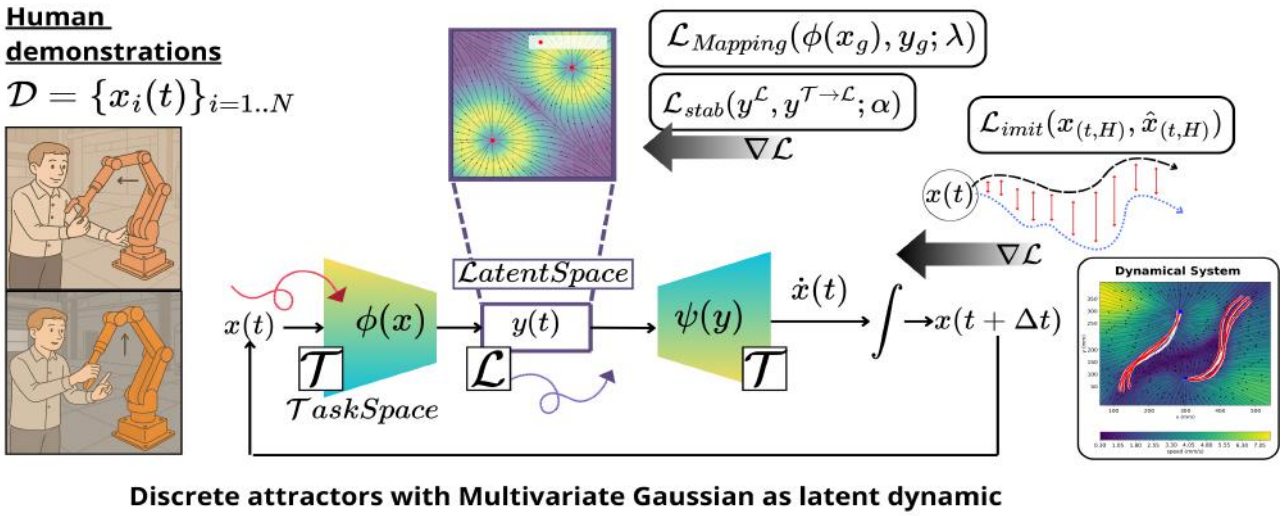


Figure 10: Architecture of the neural model based on Condor, with loss application. The illustration follow the configuration with Multivariate Gaussian and the mapping loss, with point attractors

In the first class of solutions, the research question finds an initial answer in the discrete case of attractors. In this section, we describe the methodology used with the system depicted in Figure 10, including its subcomponents (the latent dynamics and the applied loss functions). The main backbone remains the same as that of the Condor model [17], described in the previous section 1.1.3.
We refer to this implementation as **Multi-Condor**. The model's input consists of the end-effector positions from the reference demonstrations, $x(t)$, and the output corresponds to the state derivatives, $\dot{x}(t)$. Details on how these demonstrations were collected are provided in Section 2. After preprocessing, the model is trained to minimize a stability loss, $\mathcal{L}_{stability}$, alongside the imitation loss, $\mathcal{L}_{imit}$, and other custom-defined losses specific to each dynamic component.

3.1.1 Additive Dynamics

The first attempt to generalize the solution proposed for Condor with a multi-stable dynamic explored a latent model described by a summation of independent potential terms, i.e., the Gaussian potential (*Gaussian dynamic*) and the gravitational potential terms (*Gravitational dynamic*), showing that such dynamics in the 2D case exhibit the exact number of attractors that had been selected. The evolution of such dynamics in $\mathbb{R}^2$ is shown in Fig. 11:
Such a system has been constructed as follows: Consider an **additive potential function**

$$g(y) = \sum_{i=1}^{n} f(y, y_i),$$

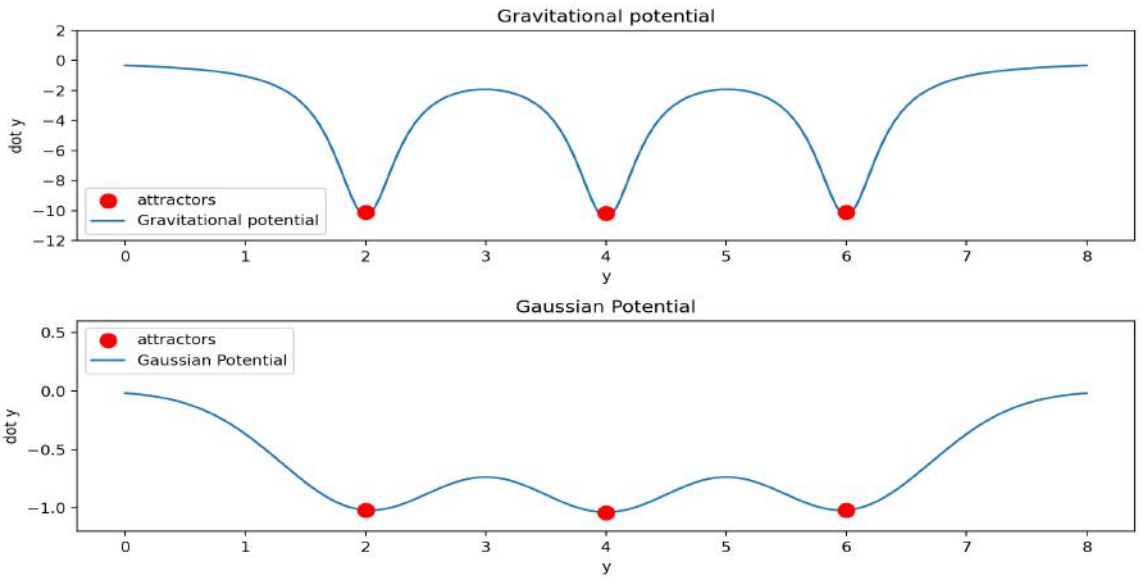


Figure 11: Comparison of the Gravitational and Gaussian potential in 2D system. In both case 3 attractors are define ($x = 2, 4, 6$)

where $f : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is continuous and almost everywhere differentiable.
We define the **vector field**

$$V(y) = \nabla g(y),$$

and require that the function g has exactly n local minima, located at the points y_i.
We enforce the following condition on the critical points of g:

$$\forall y,\ \nabla g(y) = 0 \Rightarrow y \in \{y_1, \ldots, y_n\} \vee \exists v \in \mathbb{R}^d,\ \lambda < 0 \text{ such that } H(g)(y)v = \lambda v,$$

where $H(g)(y)$ denotes the **Hessian matrix** of g at point y, and v, λ are an eigenvector and eigenvalue of that matrix.
This condition ensures that:

- The **only local minima** of g are the known points $y_1, \ldots, y_n$, and
- Any other point where $\nabla g(y) = 0$ is either a **saddle point** or a **local maximum**, because it has at least one direction of negative curvature (i.e., a negative eigenvalue of the Hessian).

This construction allows us to use the vector field $V(y)$ as a dynamical system:

$$\dot{y} = -V(y)$$

will converge only to the minima y_i, since all other critical points are unstable due to the presence of negative curvature directions. This condition is satisfied for the following systems (see Fig.11).

- **Gravitational potential**:

$$f(y) = -\frac{1}{|y - y_g|^2} \tag{4}$$

- **Gaussian potential**:

$$f(y) = -\left(e^{-(y_g - y)}\right)^2 \tag{5}$$

Although this consideration is valid, the latent state of our network has a much higher dimensionality ($y \in \mathbb{R}^{300}$). Consequently, the combination of Gravitational or Gaussian potentials results in an infinite number of attractors.

In Fig. 12, we observe the effect of this dynamic when the model is trained using only the stability loss, i.e., without considering the demonstrations. Our goal in training the models exclusively with the stability loss is to demonstrate that the system recognizes the attraction point as the equilibrium of the task dynamics. However, the system exhibits a large number of attraction points that align along a line interpolating between the two point attractors. These results suggest that the combination of multiple terms, when coupled together, generates a set of local minima that are not useful for this case study. Simulating the previously defined Gaussian dynamic in 3D space produces the landscape shown in Fig. 13. In the figure, only one attractor is defined for a system in $\mathbb{R}^3$, yet the plot clearly demonstrates that this system does not meet the problem requirements, as it presents a large number of local minima.

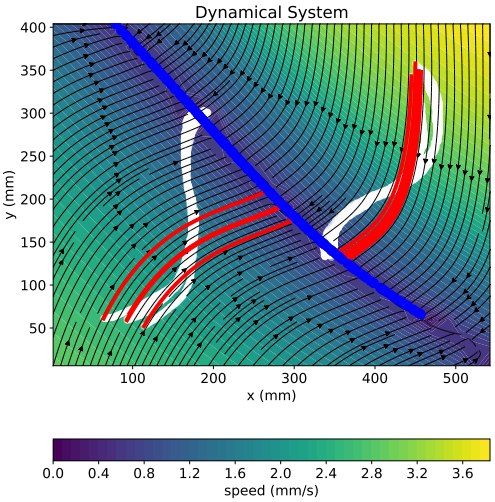


Figure 12: Stability-only output of Condor with Gaussian latent dynamics for two attractors. The blue dots represent the attractors detected by the system.

3.1.2 Partitioned Dynamic: The well-known system

The previous section demonstrates how the number of attractors increases with the system's dimensionality, resulting in an unstable solution for our point-to-point neural DMP.

To overcome this limitation, our initial approach relied on a partitioned dynamic. The first component features a single attractor and is defined as$\dot{y} = \alpha(y - y_g)$, while the second component exhibits multiple (two) equilibrium conditions. Specifically, the dynamic was defined as:

$$\begin{aligned} \dot{y}_{[:-1](t)} &= -\alpha(\hat{y}_{G0,[:-1]} - y_{[:-1]}(t)) \\ \dot{y}_{[-1]}(t) &= -\alpha(y_{g[1,-1]} - y_{[-1]}(t)) + \beta(y_{g[2,-1]} - y_{[-1]}(t))^3 \end{aligned} \tag{6}$$

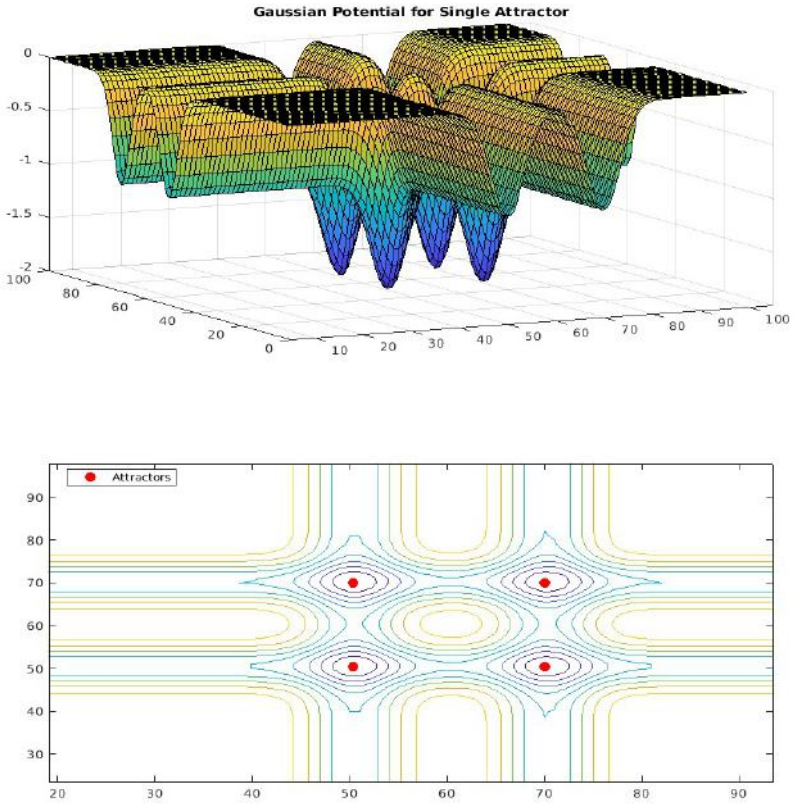


Figure 13: 3D potential landscape for a Gaussian dynamic system with a single attraction point. The plot show how increase the dimension of the state of an additive dynamics generate multiple minima

where the notation $[:-1]$ mimic the python notation, and indicate the entire state vector until the last element. Here, $\hat{y}_{G0,[:-1]}$ represents the average goal representation learned by the network. The two task goals are projected into the latent dynamic using the encoder network, and the average of these two representations is computed. For this component of the dynamic, this average goal representation serves as a single attraction point.

With the dynamic now having the correct number of attractors, our objective was to integrate the latent model within the contrastive loss framework, allowing the system to autonomously learn the appropriate representation for this goal condition.
The potentials of the two dynamic components are depicted in Fig. 14.

The presence of the exact number of attractors was empirically tested on a 4D plot, where a dynamic $\dot{y} \in \mathbb{R}^4$ was evaluated. The first three variables of the state are shown on the axes and follow the single-attractor dynamic defined in the previous section.

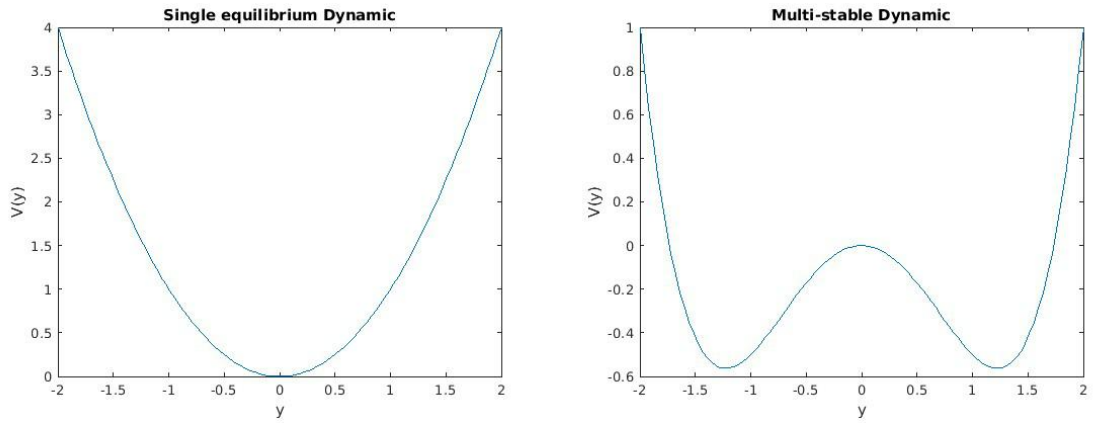


Figure 14: Potential of the 2 component of the Well-Known system. On the left the first part of the dynamic is used (single attractor). On the right, we see the multi-stable system

The fourth dimension follows the multi-stable dynamic, and its value is represented as a slice, depicted using a colormap to differentiate values.
In Fig. 15, the colormap represents the value of the fourth dimension, with darker regions indicating areas of attraction.

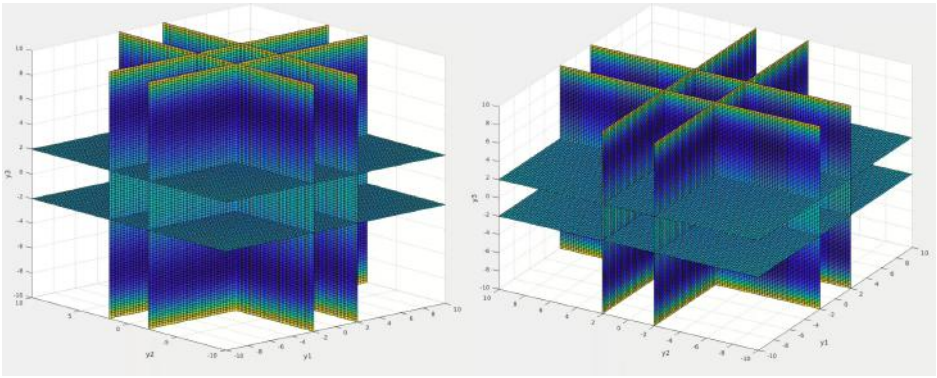


Figure 15: Well-known Potential with double attractors in a 4D dynamics, where the 4th dimension is represented as color-map. Each darker region represent a depression of the generated vector field, with a single attractor in the middle of each region

After designing the latent dynamics, we encountered a problem with the latent representation: without any additional prior, the network tends to degenerate towards a trivial solution, where all task attractors share the same latent representation. This outcome naturally minimizes the stability loss but is, of course, undesired.
To address this issue, we introduced a prior that encourages the correct representation, using a loss function composed of three distinct mean squared error (MSE) terms

$$\mathcal{L}_{well-known} = MSE(y_{g(i),[:-1]}, \hat{y}_{G(0),[:-1]}) + MSE(y_{g(1),[-1]}, -1) + MSE(y_{g(1),[-1]}, +1) \tag{7}$$

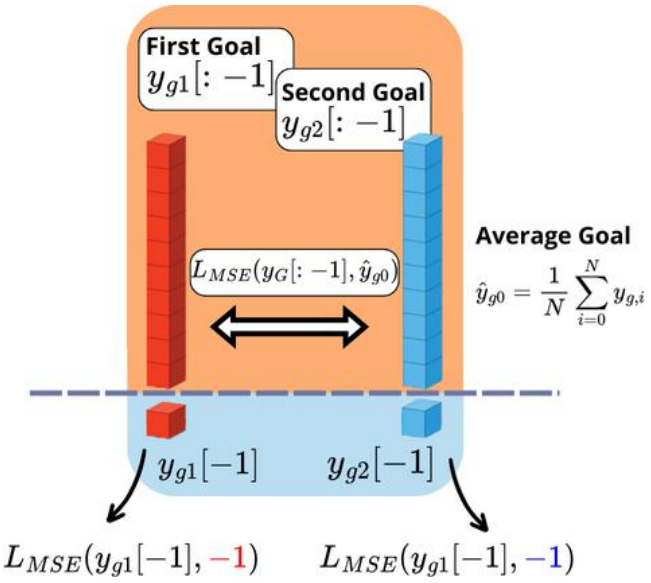


Figure 16: Scheme of the loss for mapping the goal representation of the latent dynamic to a compatible representation

This formulation aims to position the attractors of the latent system at ± 1 along the last component while ensuring a unique representation of the remaining components. The approach is illustrated in Fig. 16. This prior guide the evolution of the representation toward the desired form, enabling the system to achieve stability and be effectively used as the latent dynamic for our DMP.
The results motivate us to continue exploring the generalization to systems with increasingly larger numbers of attractors, thereby enabling the model to store multiple point-to-point trajectories in its memory. For this reason, the next section will introduce dynamic that exhibits an infinite (but fixed) number of attractors.

3.1.3 Multivariate Gaussian Dynamic, Teaching Forcing

As the final dynamic model, we evaluated a combination of multivariate Gaussian dynamics to achieve a system with a virtually infinite number of equilibrium points. The Gaussian dynamics presented in the previous section provide a smooth landscape, which facilitates the learning process of the neural model. Moreover, by coupling the i-th element of the goal state with the j-th element of the same goal state, we ensure that the dynamic maintains a fixed structure while allowing for a flexible number of equilibrium points.
The formal definition of this dynamic model is as follows:

$$\dot{y}_i = -\sum_{g=0}^{Ngoals} \{exp[\sum_{j=0}^{y_{dim}} -(y_{g,j} - y_j)^2)] * 2(y_{g,i} - y_i)\} \tag{8}$$

The exponential of the summation became the most expensive in a computational sense, but can be vectorized and computed only one time before proceed with the dynamic evaluation.
The reference vector field and his 3D version are reported in Fig 17
Although the number of attractors in the latent system is correct, the model can still collapse into a singular goal representation, converging to a random point. This phenomenon has been addressed in the field of data-driven chaotic dynamics [27]. To prevent this outcome, [27] introduced a teacher forcing approach.

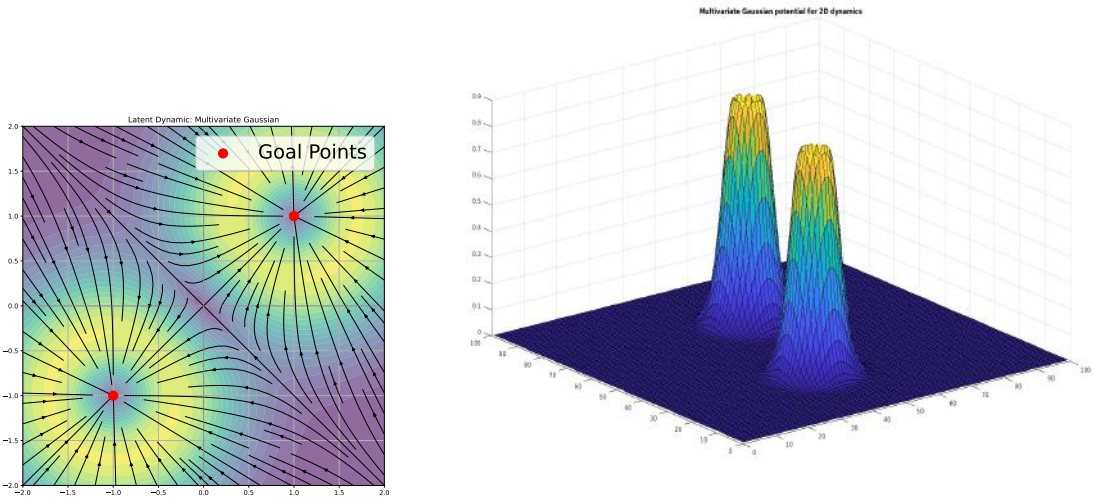


Figure 17: Vector field generated for the 2D and 3D dynamic using a multivariate gaussian dynamic

The essence of the teacher forcing concept lies in injecting the correct latent representation for the variable y during the early stages of the training process. Originally developed in the context of transformers, teacher forcing ensures that the decoder receives the correct token to learn how to decode the input signal effectively.
Similarly to our approach with the well-known dynamics in the previous section, we introduce a third loss term that maps each goal representation to a fixed reference representation, ref_g.

$$\begin{aligned} \lambda_{mapp} &= \lambda_{mapp} * \gamma \\ L(y_g) &= \lambda_{mapp} MSE(y_g, ref_g) \end{aligned} \tag{9}$$

Potentially, any latent point can be used to achieve this mapping. By introducing this loss, we provide a heuristic for the correct representation of the goals. Taking advantage of the neural network's ability to smooth the vector field, we introduce a decay rate γ for the loss weight λ_{mapp}. This approach mirrors the teacher forcing technique used in transformer models, guiding the network towards a robust representation that effectively separates the attractors. Moreover, teacher forcing has been exploit also in chaos dynamic [27]. The effect of the decay rate is visible in the experiment section, in Figure 28

3.1.4 Discrete Case: Conclusion

With this, we conclude the first section on the methodology used in this thesis. In the previous section, we explored several techniques for incorporating multi-equilibria

dynamics into our framework, enabling efficient learning of multiple and diverse demonstrations within our DMP framework. Examples of the trajectories learned using this method and comparisons are provided in the Experiments section (5).

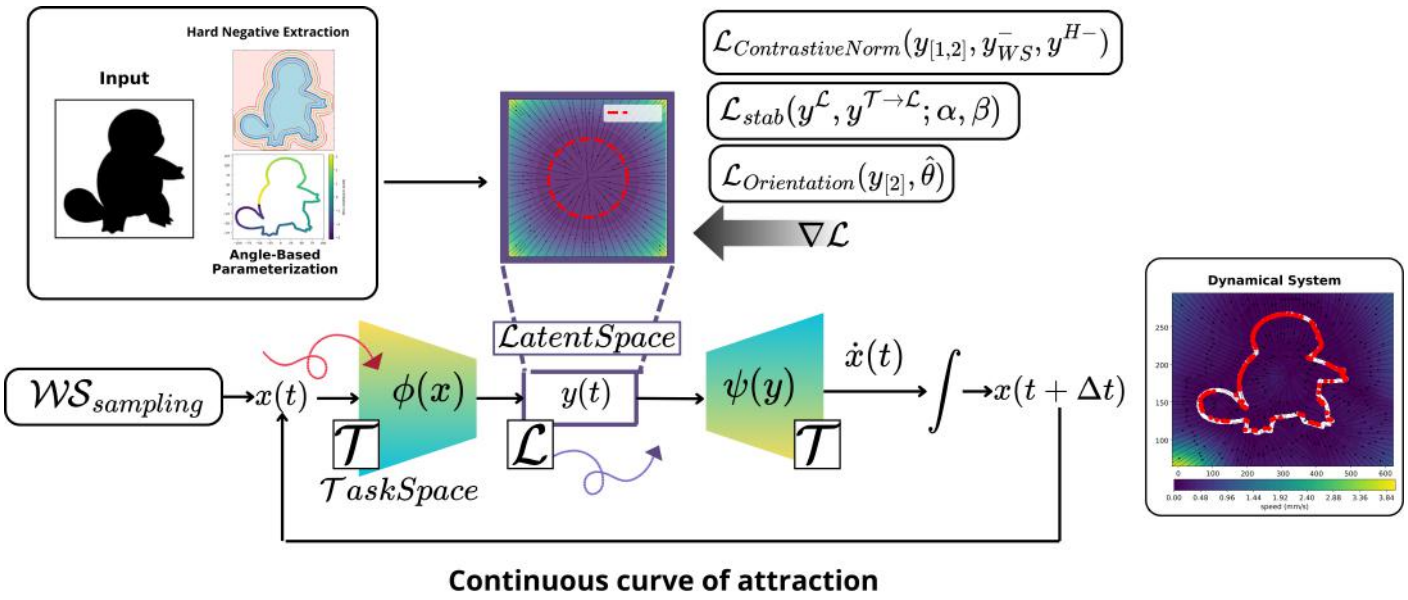


Figure 18: Architecture of the neural model based on Condor, with loss application. The illustration follow the configuration for the continuous curve of attraction

3.2 Continuous dynamic case

An extension of the concept of multi-attractor systems is the definition of a continuous curve of attraction. This approach is particularly useful in scenarios where the entire contour of an object needs to be considered as a potential endpoint of the trajectory, thereby generalizing the concept of a grasping point.

Our goal is to define a continuous shape composed of attractors. The idea is to map a set of task points forming a closed curve to specific latent points yy. These latent points are designed to satisfy certain constraints, modeled through a series of loss functions. As before, the dynamics are partitioned into two components. The shapes are extracted using standard segmentation methods, as outlined in the preprocessing section 2.2.

In the following three paragraphs, we first introduce the latent dynamics employed in our approach. Next, we discuss the conditions required to represent the task goal in a manner compatible with our imitation learning framework.

3.2.1 Continuous Dynamics

The dynamics of this system are partitioned by representing the first two components in polar coordinates, while the remaining elements are kept as in the previous setup. This transformation resembles the classical cylindrical coordinate system in $\mathbb{R}^3$. Specifically, we extract the first two elements and map them to polar coordinates using the following invertible transformation:

$$
\begin{aligned}
r &= \sqrt{y_{[1]}^2 + y_{[2]}^2} \\
\theta &= \text{atan2}(y_{[2]}, y_{[1]})
\end{aligned}
\tag{10}
$$

The goal of this component is to model a unit circle, where all points with radius $r = 1$ are considered attractors. The angular component θ remains constant over time; in other words, $\dot{\theta} = 0$. This constraint prevents the formation of limit cycles by eliminating angular drift.

The purpose of this component is to model a unit circle, where all points with a radius $r = 1$ are considered attractors. The angular component θ remains constant over time; in other words, $\dot{\theta} = 0$. This constraint effectively prevents the formation of limit cycles by eliminating angular drift.

Under this change of coordinates, the dynamics evolve as:

$$
\begin{aligned}
\dot{r} &= \beta r(1 - r)^2 \\
\dot{\theta} &= 0
\end{aligned}
\tag{11}
$$

The first equation resembles the well-known *logistic map*, modulated by the learnable parameter β. This parameter governs the system's behavior and may induce a limit cycle for values $\beta \in [3.0, 3.5)$. To ensure stable behavior and facilitate smooth learning, we model β as a function of the latent norm: $\beta_t = \phi_{|y_t|}(\|y_t\|_2^2)$ where $\phi_{|y_t|}(-)$ is a stack of Torch linear layers optimized by the stability loss. To maintain stability as discussed earlier, the value of β is clipped within the specified interval.

When applied to $y \in \mathbb{R}^2$, this polar component generates the vector field shown in Fig. 19.

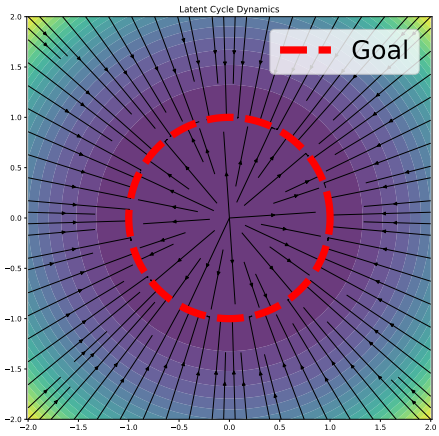


Figure 19: Vector field from the polar dynamics used for continuous 1D attractors.

The remaining dynamics (i.e., $y_{[2:]}$) can follow any single-attractor model. In our study, we observed that different behaviors emerge depending on the chosen dynamic.

First, we can let the second part of the dynamic evolve as

$$
\dot{y}_{[2:]} = -\alpha(y_{g_0} - y(t))^2
$$

where y_{g_0} represents the average of the latent points that compose the shape.

An alternative approach is to use the zero vector $\mathbf{0}$ as the attractor. In this case, the dynamic is expressed as

$$
\dot{y}_{t,[2:]} = -y_{[2:]}
$$

The results of the different dynamics are discussed in the experiments section.

Finally, we convert the polar velocity back to Euclidean space using the inverse transformation:

$$\begin{aligned} \dot{y}_{[1]} &= \dot{r}\cos(\theta) - r\sin(\theta)\dot{\theta} \\ \dot{y}_{[2]} &= \dot{r}\sin(\theta) + r\cos(\theta)\dot{\theta} \end{aligned} \tag{12}$$

The remaining components stay unchanged and do not require coordinate transformation.

3.2.2 Contrastive Norm

As in previous sections, prior knowledge is necessary to learn a meaningful latent representation. Here, we introduce a contrastive loss that enforces the attractors to lie on a unit circle within the latent space. The radial coordinate is calculated as the norm of the first two latent components:

$$r = \sqrt{y_0^2 + y_1^2}$$

By constraining the attractor points to have a unit norm in their first two components, we regularize the embedding space and prevent representational collapse.
Initially, we employed a simple mean squared error (MSE) loss to enforce the unit circle constraint. However, in the absence of counterexamples, the model exhibited a bias, projecting all points onto the unit circle. To address this issue, we adopted a contrastive loss formulation that penalizes off-curve samples differently.

$$L_{\text{contNorm}}(x) = \mathbf{1}_{\text{cycle}} \left(\|f_\theta(x)_{[:2]}\|_2^2 - 1 \right)^2 + \mathbf{1}_{\text{sample}} \left(\max\left(0, \epsilon - \left(\|f_\theta(x)_{[:2]}\|_2 - 1 \right)^2 \right) \right) \tag{13}$$

After interpolating the attractor in task space, we uniformly sample a fixed number of points. The indicator functions $\mathbf{1}_{\text{cycle}}$ and $\mathbf{1}_{\text{sample}}$ determine which component of the loss applies to each point. This framework generalizes to any continuous set of attractor points. To maximize the model's effectiveness, we employ hard negative sampling. In contrastive learning, hard negatives are samples that resemble the target class but belong to a different one. In natural language processing (NLP), this technique is used in Retrieval-Augmented Generation (RAG), where documents similar to the relevant ones are introduced

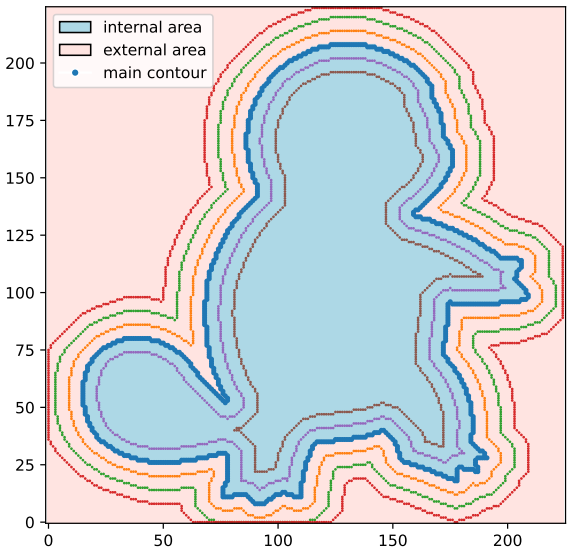


Figure 20: Hard negative extraction

as challenging contrasts, forcing the model to maximize the distance between similar but distinct samples.
In our context, hard negatives are sampled near the attractor manifold. For each batch, we sample points from three distinct sets:

- Points on the attractor manifold, y_g
- Nearby points (hard negatives), $\tilde{y}_g$
- Random points sampled from the full workspace

This strategy consistently enhances model performance and is used as the default setup in our experiments. An example of hard negative sampling is shown in Fig.20.

3.2.3 Orientation mapping

A second limitation we encountered during training was the overrepresentation of certain segments of the attraction curve, which led to disproportionately strong attraction toward specific regions of the continuous attractor.
To address this, we introduced a second term in the loss function. Since each point along the task curve is mapped onto a unit circle in the latent space, this term ensures that each point is assigned a specific location on the circle. The mapping is parameterized by the radius (as handled by the first loss term) and the orientation of each point.
As described in the preprocessing section 2.2, points on the task curve are first sorted, and an orientation value $\hat{\theta}_i \in [-\pi, +\pi]$ is assigned to each. This additional constraint encourages a more uniform distribution of the generated attraction across the latent space, thereby improving the model's overall performance.
The choice of distance metric in this mapping poses an interesting challenge for geometry learning. The discontinuity introduced by extreme elements of the domain creates issues when using a simple Euclidean distance function, such as $\mathcal{L} = \text{MSE}(\hat{\theta}, \theta_d)$, which can lead to unstable training.
A well-established technique to address such singularity issues involves transforming the angle using trigonometric functions. Several works [[28],[29]] apply a transformation of the form $z(\theta) = (\sin(\theta), \cos(\theta))^T$. This representation maps the extreme cases, such as $\theta = 360°$ and $\theta = 0°$, to the same point, correctly capturing the fact that they represent the same orientation.
Using this idea, the loss function can be reformulated as:

$$z(\theta) = [\sin(\theta), \cos(\theta)]$$

$$\mathcal{L} = \text{MSE}\left(z(\hat{\theta}), z(\theta)\right)$$

A second approach relies on a contrastive formulation of the same loss, which has been evaluated using both the original angular representation θ and the trigonometric

representation $z(\theta)$. Given the natural distribution of points along the task's attraction curve, we adopt a contrastive loss function that accounts for the noisy outputs produced by the segmentation process. Specifically, the loss encourages similarity between corresponding representations while tolerating small deviations, and is defined as:

$$\mathcal{L} = \max\left(0, \epsilon - \text{MSE}(z(\hat{\theta}), z(\theta_d))\right)$$

An evaluation and discussion of the different loss formulations are provided in Section 4.

3.2.4 Continuous Case: Conclusion

With the proposed approach, we successfully inferred a specific shape from an object, which was then used as an attraction contour in the task space. The method demonstrated its reliability across various tasks, including both simple geometric shapes and more complex contours extracted from stylized or irregular images.
Furthermore, once the object was segmented, we observed stable convergence toward the set of points defining the task curve. This provides evidence that the proposed method effectively supports the generation of consistent attraction dynamics from segmented object geometry.

3.3 Manifold dynamics: Puma

The architecture of Condor [17] constrains the latent dynamics to evolve according to a specific hard-coded function, $\dot{y} = f^{\mathcal{L}}(y)$. By removing this constraint and replacing it with a numerical gradient-based formulation of the function $f^{\mathcal{L}}$, the system can be tuned around the concept of distance metrics. These selected distance metrics model the latent space as a manifold, allowing it to represent different geometric spaces (see Fig.21).

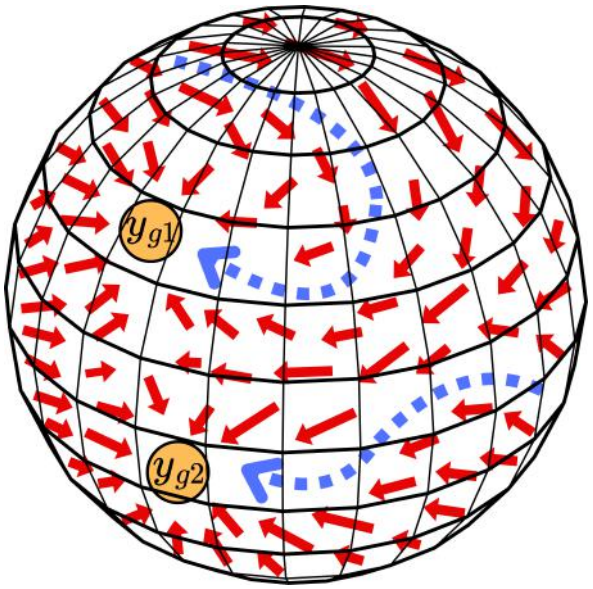


Figure 21: Example of trajectories on Spherical space

With these additional considerations, Condor naturally evolves into Puma [18]. This new framework relaxes the original loss formulation and redefines the embodiment of contrastive learning, transitioning from a contrastive loss to a triplet loss.
To integrate Puma within a multi-attractor setting, we introduce a decision function that selects the nearest goal point for each sample y. In the following section, we provide a detailed review of the chosen distance metrics and the updated loss functions.

3.3.1 Gradient-based latent dynamic

In both the discrete and continuous cases, the models learn a Cartesian vector that describes the position of the end-effector. Learning the orientation of a given trajectory, however, introduces a new challenge for our framework. Puma [18] addresses this problem by modeling latent dynamics on different manifolds. The previously hard-coded latent dynamics are replaced with a numerical expression, resulting in:

$$\dot{y}(t) = \frac{\partial \psi(x)}{\partial x} \tag{14}$$

Learning the evolution of a dynamical system as the Jacobian of a direct mapping is a well-established paradigm, explored in several works such as [19]. In contrast to most prior approaches, Puma does not impose constraints on the gradient of the encoder. Other methods often require the learned function to be semi-positive (or semi-negative) definite, or to be invertible.
The flexibility of this formulation—free from strict structural constraints—offers a significant advantage: it allows seamless integration into a wide range of neural architectures without limiting the expressiveness of the function space.

The integration scheme remains the same as in Condor, where the evolution of both the task trajectory $x(t)$ and the latent trajectory $y(t)$ is computed using a multi-step approach, relying on backpropagation through time.

3.3.2 Distance Metrics and loss formulation

In contrast to Condor[17], the stability loss in our framework is relaxed. Specifically, we adopt a triplet loss, as described in the introduction, which maximizes the distance from previously visited states while minimizing the distance to the goal (see Equation 17). A similar triplet loss formulation was also used in the Condor framework under the name *Relaxed-Condor*.
The key difference from that approach lies in the choice of distance metric, which in our case accounts for the underlying different geometry of the latent space. Examples of such distance functions $\delta(-,-)$ include the great-circle (spherical) distance and the Euclidean distance. The second is define as

$$\delta_e(a,b) = \|a-b\|_2 \tag{15}$$

while the spherical distance is based on the cosine similarity between 2 vectors (a,b) :

$$\cos_{simil}(\theta) = \frac{a \cdot b}{\|a\|\|b\|}$$

$$\delta_s = arcos(cos_{sim}(a,b)) \tag{16}$$

With the upper define distance metrics, the stability loss in the spherical domain takes the form of

$$\mathcal{L}_{stable} = \sum_{b=0}^{B-1} \sum_{t=1}^{H} max(0, m + \delta_s(y_{g_i}, y(t+\Delta t)) - \delta_s(y_{g_i}, y(t))) \tag{17}$$

.
A solution to extend the loss formulation to multiple point attractors involves assigning different end points y_{g_i} to each sampled point $y(t)$.
In our experiments, we evaluated three different selection functions. Each function takes as input a point in the workspace and returns the nearest attractor.
The first option relies on the task-space distance between the sampled point and the task attractors:

$$d^{\mathcal{T}}_{selection}(x) = min_{x_{g_i}}(\delta(x, x_{g_i}))$$

In this case, the issues encountered in the previous section resurface. In particular, the contrastive learning approach does not provide an interpretable mapping for each task point, meaning there is no guarantee of the validity of the nearest task attractor once the points are projected into the latent space.

A second solution addresses this problem by moving the selection function directly into the latent space:

$$d^{\mathcal{L}}_{selection}(y) = min_{y_{g_i}}(\delta(y, y_{g_i}))$$

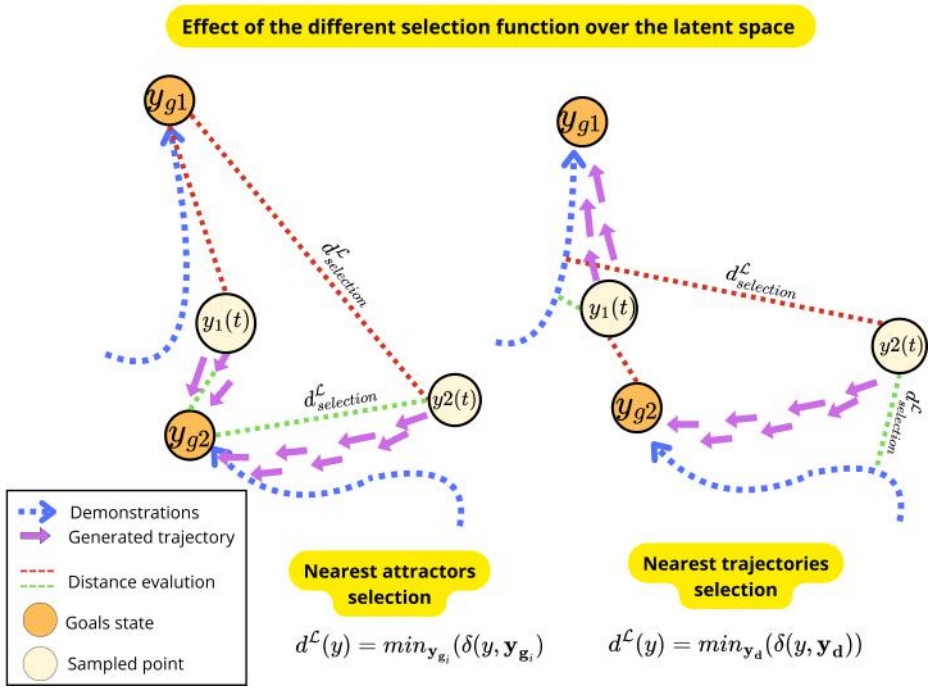


Figure 22: Effect of the different attractors selection function, on the left figure we can see a case in which the point $y_1(t)$ is very close to the first trajectory, but the attractor distance guide the particle toward the wrong goals. In the second case, we see how the computation of the nearest trajectory can separate the space

This approach introduces a strong bias toward the goals, but it can hinder the learning process when trajectories pass close to each other (see Fig.22). If, at some point, a sample point y from a specific demonstration is near a different goal, the stability loss formulation will force the system to evolve according to the other solution.
To address this issue, we introduce our final formulation for the selection function. Instead of considering the nearest attractor y_{g_i}, we consider the nearest point on the reference trajectories y_t. Once we have established which demonstration is closest to our sample, we can select the goals of that trajectory as attractors for the point y via the mapping $G(y_t)$.
This procedure can be described with the following equation:

$$y_t = min_{y_d}(\delta(y, y_d)) \tag{18}$$

$$y_g* = G(y_t) \tag{19}$$

Efficient handling of the distance selection function relies on a KD-Tree implementation, which constructs a tree of the demonstration points and provides a time-efficient solution for finding the nearest neighbors.

3.3.3 Boundary Loss

The performance of both models begins to degrade as the dimensionality of the workspace increases, particularly when approaching the workspace boundaries. The boundary loss is designed to ensure that the dynamic system's states remain within the prescribed workspace, defined as a hypercube with each dimension ranging from $[-1, 1]$. To achieve this, the loss function enforces that the velocities at the boundary do not point outward, thereby preventing the system from leaving the workspace. Specifically, for each dimension i of the state space, we consider two boundary conditions: the **upper boundary** at $x_i = 1$ and the **lower boundary** at $x_i = -1$.
The method first identifies the states that lie close to the boundaries by computing the distance between the current state and the boundary values. For the upper boundary, this distance is given by:

$$\text{distance}_{\text{upper}} = |x_i - 1|$$

and for the lower boundary, it is:

$$\text{distance}_{\text{lower}} = |x_i + 1|$$

If the distance is less than a small threshold ϵ, the state is considered to be at the boundary.
To assess whether the velocity at the boundary points inward or outward, the method computes the **dot product** between the velocity vector $\mathbf{v}$ at the boundary and a **normal vector** representing the inward direction. The normal vectors are defined as:

$$\mathbf{n}_{\text{upper}} = [0, \ldots, 1, \ldots, 0] \quad \text{and} \quad \mathbf{n}_{\text{lower}} = [0, \ldots, -1, \ldots, 0]$$

The dot products are then calculated as:

$$\text{dot product}_{\text{upper}} = \mathbf{v} \cdot \mathbf{n}_{\text{upper}} \quad \text{and} \quad \text{dot product}_{\text{lower}} = \mathbf{v} \cdot \mathbf{n}_{\text{lower}}$$

A **positive dot product** indicates that the velocity vector is pointing **outward**, while a **negative dot product** indicates an **inward** motion. Since only the outward motion should be penalized, the loss function applies a **ReLU activation** to the dot products, defined as:

$$\text{loss} = \frac{1}{2 \times d} \sum_{i=1}^{d} \Big[\text{ReLU}(\text{dot product}_{\text{upper}}) + \text{ReLU}(\text{dot product}_{\text{lower}}) \Big]$$

Here, d is the dimension of the workspace. The ReLU function ensures that only positive dot products contribute to the loss, thus preventing outward motion at the boundaries. Finally, the loss is averaged across dimensions and weighted by a hyperparameter λ to control its relative importance. This formulation effectively constrains the system to remain within the workspace while allowing freedom of movement within the limits.

3.3.4 Orientation Learning: conclusion

With this, we terminate also the integration of the multi-attractor use case with Puma. With this, we was able to address both position and orientation of our robot end-effector, by leveraging different distance metrics and substitute the desired attraction point with the nearest attractor, following the decision strategy presented before.

4 Evaluation Strategy

A first classification of the metrics used in this thesis divides them into **stability metrics** and **imitation metrics**. As the names suggest, stability metrics assess the model's ability to exhibit stability during training and its speed of convergence. In contrast, imitation metrics evaluate how faithfully a model can replicate a given task, such as trajectory imitation or shaping learning.
In the following section, we will first introduce the metrics used in both the discrete and continuous settings. After that, we provide a complete overview of the benchmark models used during the evaluation phase. Finally, the results section presents both a qualitative analysis—examining the vector fields generated by the different models — and a quantitative comparison based on the various metrics.

4.1 Metrics

A second division for our metrics, following the same division used in the methodology section, is about the metrics used in the discrete case and in the continuous case.

4.1.1 Metrics for discrete attractors

In the discrete setting, our model is expected to replicate multiple point-to-point trajectories while exhibiting specific stability behaviors.
To evaluate imitation performance, we utilize the following metrics: the **Fréchet Distance**, the **Dynamic Time Warping Distance** (DTWD), and the standard **Root Mean Square Error** (RMSE).
To assess **stability**, we focus on the **number of spurious attraction points**. We begin by sampling a set of initial conditions from a uniform grid. For each point, we let the system evolve until convergence. We then examine the resulting attractors: if the distance between an attractor and the nearest goal exceeds a fixed threshold, it is labeled as a *spurious attractor*. Additionally, we report the **average distance to the goals** across all converged trajectories, which reflects the precision of the vector field.
We'll now proceed to analyze the properties of each imitation metric.
The **Dynamic Time Warping** (DTWD)[30] has been widely used to compare time series, especially in speech recognition. Unlike Euclidean distance, which compares values at identical time indices, DTWD accounts for variations in time alignment, allowing flexible matching of sequences that may be misaligned in time.
Given two time series

$$x = (x_0, x_1, \ldots, x_{n-1}) \quad \text{and} \quad x' = (x'_0, x'_1, \ldots, x'_{m-1}),$$

we define a local distance function $d(x_i, x'_j)$, typically the Euclidean norm. A *warping path* π is a sequence of index pairs

$$\pi = \big((i_0, j_0), (i_1, j_1), \ldots, (i_L, j_L)\big),$$

satisfying:

- **Boundary conditions**: $(i_0, j_0) = (0, 0)$, $(i_L, j_L) = (n-1, m-1)$,
- **Monotonicity**: $i_{k+1} \geq i_k$, $j_{k+1} \geq j_k$,
- **Step size**: $(i_{k+1} - i_k, j_{k+1} - j_k) \in \{(1,0), (0,1), (1,1)\}$.

The DTW distance of order q is given by:

$$\mathrm{DTWD}_q(x, x') = \min_{\pi \in \mathcal{A}(x,x')} \left(\sum_{(i,j)\in\pi} d(x_i, x'_j)^q \right)^{1/q},$$

where $\mathcal{A}(x, x')$ denotes the set of all admissible warping paths.
the DTWD has the following properties:

- Non-negativity: $\mathrm{DTWD}_q(x, x') \geq 0$
- Identity: $\mathrm{DTWD}_q(x, x) = 0$
- Symmetry: $\mathrm{DTWD}_q(x, x') = \mathrm{DTWD}_q(x', x)$

However, DTWD does **not** satisfy the triangle inequality, and hence it is a *semi-metric*. Its main advantage lies in its temporal invariance, allowing for accurate alignment of patterns that may be stretched or compressed in time.
The second metric used in this comparison is the **Fréchet Distance** [31]. This metric also measures the similarity between two curves by considering both the location and the ordering of points along the curves.
Let $f, g : [0, T] \to \mathbb{R}^d$ be two continuous curves. The Fréchet distance $d_F(f, g)$ is defined as:

$$d_F(f, g) = \inf_{\alpha,\beta} \max_{t\in[0,T]} \|f(\alpha(t)) - g(\beta(t))\|,$$

where $\alpha, \beta : [0, T] \to [0, T]$ are continuous, non-decreasing reparameterization of the interval.
This metric satisfies all the properties of DTWD, and additionally the **triangle inequality**. Therefore, the Fréchet distance is a valid *metric*, making it a reliable similarity measure in our context.
Finally, we compute the **RMSE** between the demonstration trajectories and the generated ones. This is the simplest of the three metrics and measures the quadratic error using the Euclidean distance (i.e., the squared norm). Given two aligned trajectories $x = (x_1, \ldots, x_n)$ and $\hat{x} = (\hat{x}_1, \ldots, \hat{x}_n)$, the RMSE is defined as:

$$\mathrm{RMSE}(x, \hat{x}) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \|x_i - \hat{x}_i\|^2}.$$

RMSE assumes identical parametrization of the curves and does not account for any reparametrization or temporal misalignment.

Stability Metrics — As introduced earlier, we use the number of spurious attractors to assess the smoothness of the vector field. A set of initial conditions is sampled from a uniform grid, and each trajectory is evolved for 500 time steps with a fixed time increment Δt. The final state of each trajectory is analyzed: if its distance from the closest goal exceeds a fixed threshold (set to 0.5 pixels in our experiments), it is marked as a *spurious attractor*.

Additionally, we compute the **average distance to the goals**, which serves as a measure of precision. A system with few spurious attractors but a large average distance may still fail to accurately reach the goals. Therefore, both indicators are necessary to assess whether the system converges reliably and with sufficient precision.

4.1.2 Metrics for continuous attractors

In the continuous setting, our primary focus shifts from trajectory imitation to analyzing the **stability properties of the diffeomorphisms**. Here, we are less concerned with reproducing the exact shape of the demonstration trajectories and instead emphasize the fidelity of the *attraction curve* with respect to the *original contour*.

To evaluate the similarity between the goal contour and the final converged trajectories, we adopt the **Chamfer Distance** [32], a metric commonly used in point cloud alignment and shape matching.

Let $X = \{x_1, x_2, \ldots, x_N\} \subset \mathbb{R}^d$ and $Y = \{y_1, y_2, \ldots, y_M\} \subset \mathbb{R}^d$ be two point sets representing the reference and generated contours, respectively. The asymmetric Chamfer Distance from X to Y is defined as:

$$d_{\text{Chamfer}}(X, Y) = \frac{1}{|X|} \sum_{x \in X} \min_{y \in Y} \|x - y\|^2,$$

and the full (symmetric) Chamfer Distance is:

$$\text{CD}(X, Y) = d_{\text{Chamfer}}(X, Y) + d_{\text{Chamfer}}(Y, X).$$

This metric penalizes the average nearest-neighbor distance between points across both point sets, capturing both completeness and accuracy.

The contour of the original shape is extracted through a preprocessing phase that returns a variable number of points, typically in the order of thousands. To compare the generated and reference contours efficiently, we must compute nearest-neighbor distances between each point in one cloud to those in the other.

To accelerate this computation, we use a **KD-Tree** data structure to index both the point sets. A KD-Tree partitions the space hierarchically, enabling nearest-neighbor queries in $O(\log n)$ time per point, resulting in an overall search complexity of $O(n \log n)$, as opposed to the naive $O(n^2)$ search.

This is especially important because, although the evaluation of Chamfer Distance is typically performed once per training iteration or experiment, efficient implementations are crucial when optimizing the code for large-scale evaluations or real-time feedback.

Stability in the Continuous Case — Beyond the shape fidelity, we are interested in whether the continuous dynamics defined by the diffeomorphism converge smoothly to the target contour. As in the discrete case, we can analyze the number of **spurious attractors**, i.e., points of convergence that lie outside the contour boundary. Alongside, this setting require to map each point on the target curve to a specific point on the latent unit circle, so a measure of the distance between the angular value predicted from the encoder and the one assigned during the preprocesing section is used for analyze the effect of the different losses on the latent space manifold.
In this context, the Chamfer Distance provides a global, measure of how well the vector field preserves the geometry of the original shape, making it a natural substitute for imitation-based metrics used in the discrete setting. While the stability metrics offer a overview of the precision of the solution and on the stability of the training procedure

4.2 Benchmark

Provinding benchmark for our task results difficult, as when we start to study this topic we was the first to address the problem of multi-stable motion primitive.
The only work that we found on the the topic was [5], that utilize a supported vector machine to partition the vector field and having N area where each area offer a unique attractors. No implementation of the methods described in that paper has been found.
An alternative has been published once that our work has started, and it's represented from the work of Sochopoulos et. al. [8], a lyapunov-based learner that can address N different attractors, named **StableNODE**. With the kind help of the authors, that share with us their code, we implement the solution proposed on our custom dataset and use it for benchmarking our results in the Discrete Attractor case.
The *StableNODE* is defined as follows:

$$\begin{aligned} x(t_0) &= \phi_\theta(y), \\ \dot{x}(t) &= \hat{f}(x), \quad t \in T, \\ z(t) &= \psi_\theta(x(t)), \end{aligned}$$

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

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

where $L(x) = \nabla V_\theta(x) f_\theta(x) + \alpha V_\theta(x)$, and ϵ, α, s are positive scalars. $V_\theta(\cdot)$ is a candidate Lyapunov function.

The rest of the notation aligns with that used in Definition 1. Importantly, even though the Lyapunov network and the mappings for input, output, and state are represented by distinct networks, they are all trained together by optimizing the Neural ODE loss. The purpose of the modified dynamics $\hat{f}$ is to reproduce the nominal dynamics f_θ in regions where $L(x) \leq 0$. This condition implies that the nominal dynamics naturally lead to exponential stability with respect to the Lyapunov function V_θ, thus requiring no intervention. In contrast, when $L(x) > 0$, the dynamics are corrected using the control input $u(x)$ to enforce stability. Both ϕ_θ and ψ_θ are constructed as bijective layers ($\psi_\theta = \psi_1 \circ \psi_2 \circ \psi_3 ...$). Each layers is implemented in the form of

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

as describe in [33].

More over, in the following section the reader will find a comparison with a trivial methods, corresponding to the stack of N single-attractors Condor model. Each model is on charge of his sub-area, partitioning the space using a distance function from the attractors itself. Such solution can work only in simple case, where the domain of each demonstrations is well separated from the rest.

Both methods suffer from the vicinity of different demonstration, and loose their stability property when this case is encounter. In the case of the stack of single-attractor Condor, the limit of each domain offer an unstable behavior, while the summation of several lyapunov terms (one for each attractors) can results in a conflictual behaviour

5 Experiment

In this chapter, we present the results of several experiments. We begin by reviewing the developed solution for the discrete case, analyzing the effect of the latent dynamics and the contribution of the different loss terms. Next, we evaluate the results for the continuous attraction curve case, again examining the influence of each dynamic component and the role of the loss terms. Finally, we explore the real-world experiments conducted with an industrial manipulator—the KUKA iiwa14, equipped with a soft hand provided by QB-Robotics.

5.1 Discrete Case

The stability loss provides conditions under which the neural model can map an unknown dynamic to a latent dynamic in a stable manner, by constructing a diffeomorphism between the task space X and the latent space Y. As discussed in the previous section, this loss alone is insufficient to produce the correct mapping when the latent dynamics exhibit multiple attractors.
In the case of the well-known system, a mapping loss is required to associate the two goals (y_{g_1}, y_{g_2}) with fixed latent points $(-\mathbf{1}, +\mathbf{1})$. This loss is implemented as a mean squared error between the reference points and the corresponding target vectors.

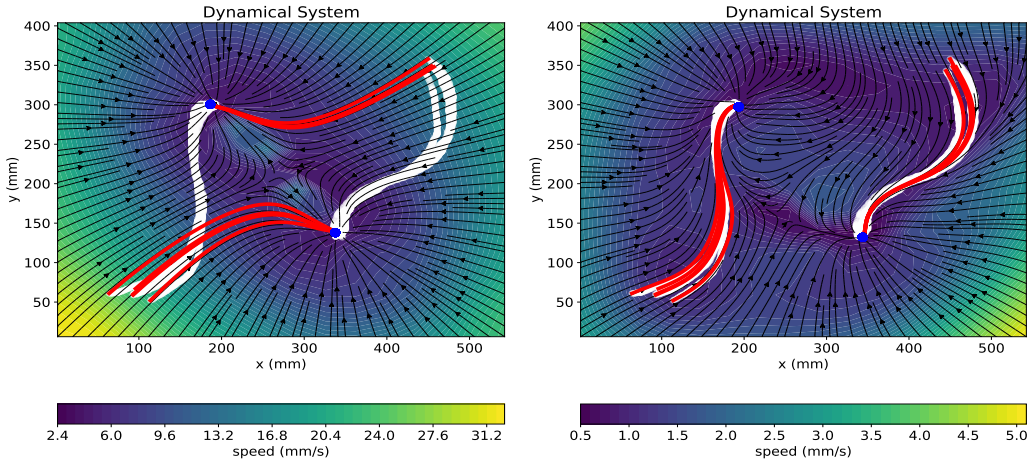


Figure 23: Results of the Well-known dynamic (composed of a single attractor system $\dot{x}_{[:-1]} \in \mathbb{R}^{299}$ and a second system that exhibit a double equilibrium $\dot{x}_{[-1]} \in \mathbb{R}$) **(A)** The system learns the stable vector field by mapping the task space to latent dynamics, without considering trajectories. **(B)** Same setup as (A), but with imitation loss added to mimic reference trajectories (white), while the red trajectories are generated by the model.

Results for the two-attractor case—using a combination of two losses (stability loss and mapping loss), and three losses (stability loss, mapping loss, and imitation loss)—are shown in Fig 23.
When transitioning to the multivariate Gaussian dynamics, we encounter a similar

challenge. If the latent representations of the goals collapse into a single point, we observe the same effect as in the well-known dynamics without the mapping loss term. Conversely, enforcing fixed latent representations for the goals in the multivariate Gaussian setup results in a strong partitioning of the task space, as shown in the first row of Fig. 24. Initially, this causes low-velocity regions, which gradually diminish as the network learns appropriate embeddings for the task attractors. Over time, the stability loss smooths the vector field. The effect of the teaching-forcing approach is visible in the second row: once the influence of the mapping loss fades, the network has already learned a consistent representation of the task attractors, leaving the stability loss as the primary factor shaping the generated vector field.

Table 1: Metrics for the discrete attraction case

2 Attractors						
Method	**RMSE (mm)**	**DTWD (mm)**	**FD (mm)**	**Spurious Attractors percentage**	**Distance From Goals (px)**	**stable?**
multiple single-attractor Condor	54.29 ± 38.05	60.77 ± 58.53	99.62 ± 87.04	68.31%	32.29 ± 38.96	X
stableNODE	30.01 ± 26.24	24.00 ± 23.17	67.33 ± 60.11	76.95%	96507.60 ± 61969.44	✓
Multi-Condor (only BH)	9.47 ± 6.34	3.14 ± 1.14	12.50 ± 4.40	39.35%	29.91 ± 12.35	X
Multi-Condor (Stability loss and T.F.)	154.69 ± 48.50	82.04 ± 22.09	145.70 ± 48.92	0.00%	1.06 ± 0.25	✓
Multi-Condor (No T.F.)	12.39 ± 4.68	4.13 ± 0.66	11.92 ± 0.89	1.62%	3.35 ± 2.42	X
Multi-Condor	$\mathbf{9.76 \pm 6.55}$	$\mathbf{3.29 \pm 1.11}$	$\mathbf{9.92 \pm 3.81}$	**0.00%**	$\mathbf{3.74 \pm 1.2}$	✓
3 Attractors						
Method	**RMSE (mm)**	**DTWD (mm)**	**FD (mm)**	**Spurious Attractors percentage**	**Avg Distance From Goals (px)**	**stable?**
multiple single-attractor Condor	84.70 ± 24.53	99.84 ± 41.36	147.79 ± 62.19	69.30%	66.36 ± 26.56	X
stableNODE	26.18 ± 6.85	19.50 ± 7.91	57.17 ± 18.18	97.04%	117042.63 ± 41359.19	✓
Multi-Condor (only BH)	15.60 ± 4.21	5.69 ± 0.73	16.65 ± 3.02	62.81%	31.95 ± 14.19	X
Multi-Condor (Stability loss and T.F.)	110.30 ± 23.28	110.12 ± 51.60	151.90 ± 163.97	0.00%	1.53 ± 0.54	✓
Multi-Condor (No T.F.)	19.06 ± 5.82	8.34 ± 8.29	21.28 ± 7.52	5.10%	6.76 ± 2.63	X
Multi-Condor	$\mathbf{16.30 \pm 4.48}$	$\mathbf{5.69 \pm 0.96}$	$\mathbf{16.81 \pm 2.93}$	**0.00%**	$\mathbf{2.74 \pm 0.93}$	✓

Given the advantages of the multivariate Gaussian dynamics combined with the teacher forcing strategy—namely, the ability to handle a virtually infinite number of attractors—we selected this configuration as our final implementation for the discrete case. In Table 1, we present a quantitative analysis against our benchmark, along with an evaluation of the contribution of each individual loss component on datasets containing 2 and 3 attractors. The reported metrics are averaged over five random seed initializations for each dataset, using a total of four datasets per configuration.

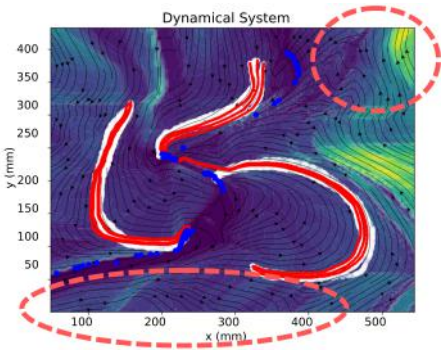


Figure 25: Velocity profile generated using the StableNODE approach [8]. The dashed line highlights how, before reaching the equilibrium point, the vector field drives the particle away—sometimes even beyond the workspace boundaries

Starting with the stability metrics, we recall that they are computed empirically: the workspace is uniformly sampled, and each particle's position is updated based on the output of the neural model. In the case of StableNODE, the generated vector field drives the particles away from the workspace boundaries (see Fig. 25). However, after the fixed number of steps we define for evaluation, the particles often remain far from the target points. Although

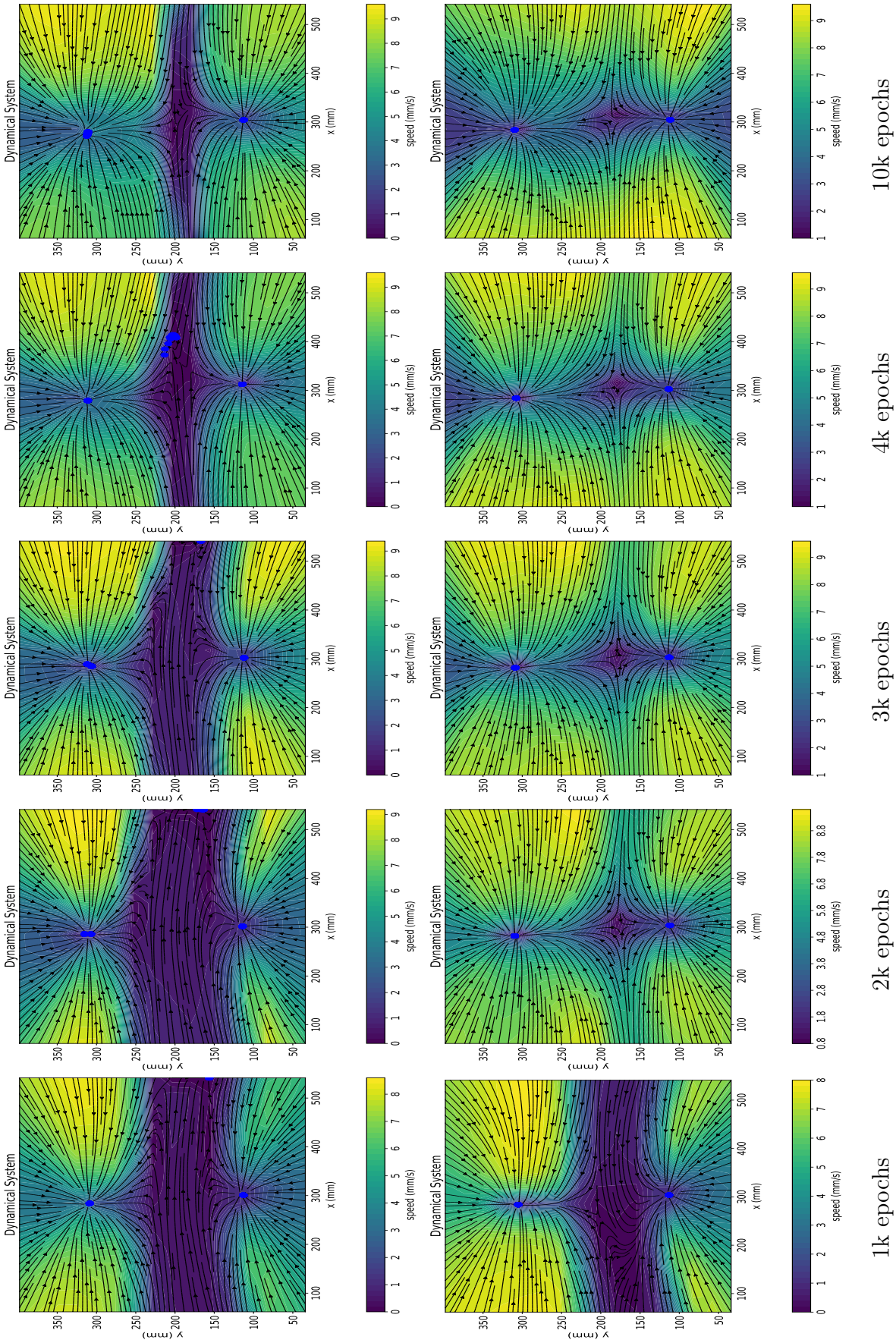


Figure 24: Effect of the Teacher Forcing approach on the evolution of the generated vector field. The latent dynamics are defined by a multivariate Gaussian, and the training incorporates both the stability loss and the mapping loss, each with different decay rates. **A**: Evolution for $\gamma = 1$ (no decay of the mapping loss). **B** : Evolution for $\gamma = 0.99$ (mapping loss decays over times).

the system is stable by design, its velocity profile prevents convergence to the goals within the selected number of steps 25. The threshold ϵ used for the compute the percentage of spurious attractors was 5 px.

Moreover, we can evaluate the impact of teacher forcing from a numerical standpoint. Although the imitation metrics are not significantly influenced by the introduction of this loss, the overall system becomes more stable. In particular, when transitioning from two to three attractors, the benefit of incorporating these additional terms becomes evident.

Using only the stability losses in conjunction with the teacher forcing approach yields the lowest stability metrics among the evaluated methodologies, highlighting the advantages introduced by these terms when we learn a diffeomorphism.

On the next figure (Fig. 26), we present different vector fields generated by the StableNODE architecture, the if-else methodology, and our multivariate solution. It is clear that the proposed method produces the smoothest vector field while exhibiting the fewest spurious attractors.

Finally, on the following page (Fig. 27), we present a set of demonstrations solved using the Multivariate Latent Dynamics model combined with the teacher forcing approach.

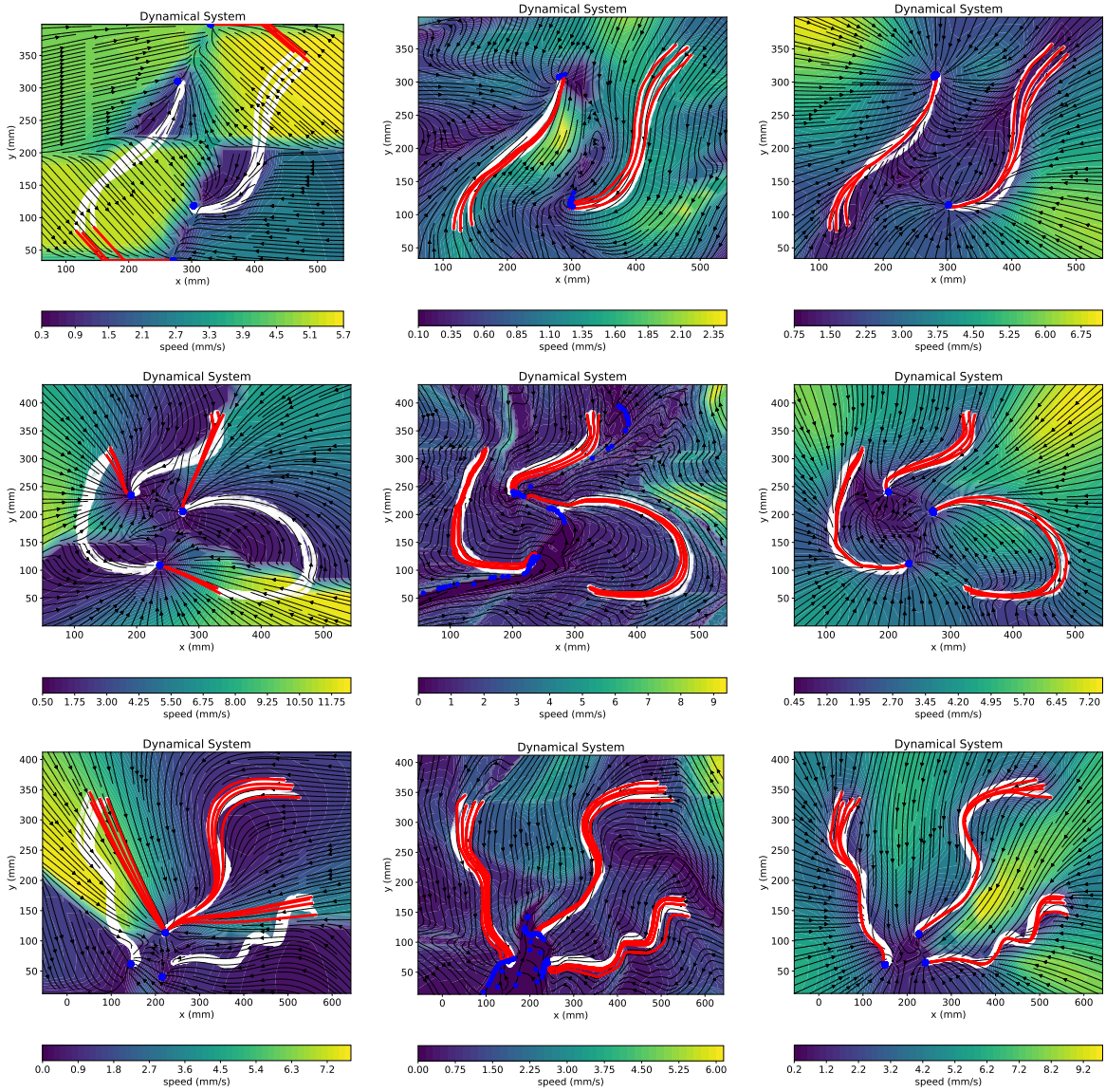


Figure 26: Vector fields generated with different methodologies.
The first column are obtained with the stack of single attractor condor, the second column represent the output of the StableNODE system. Finally, the last column represent the output of the Multi-Condor with Multivariate Gaussian latent dynamic

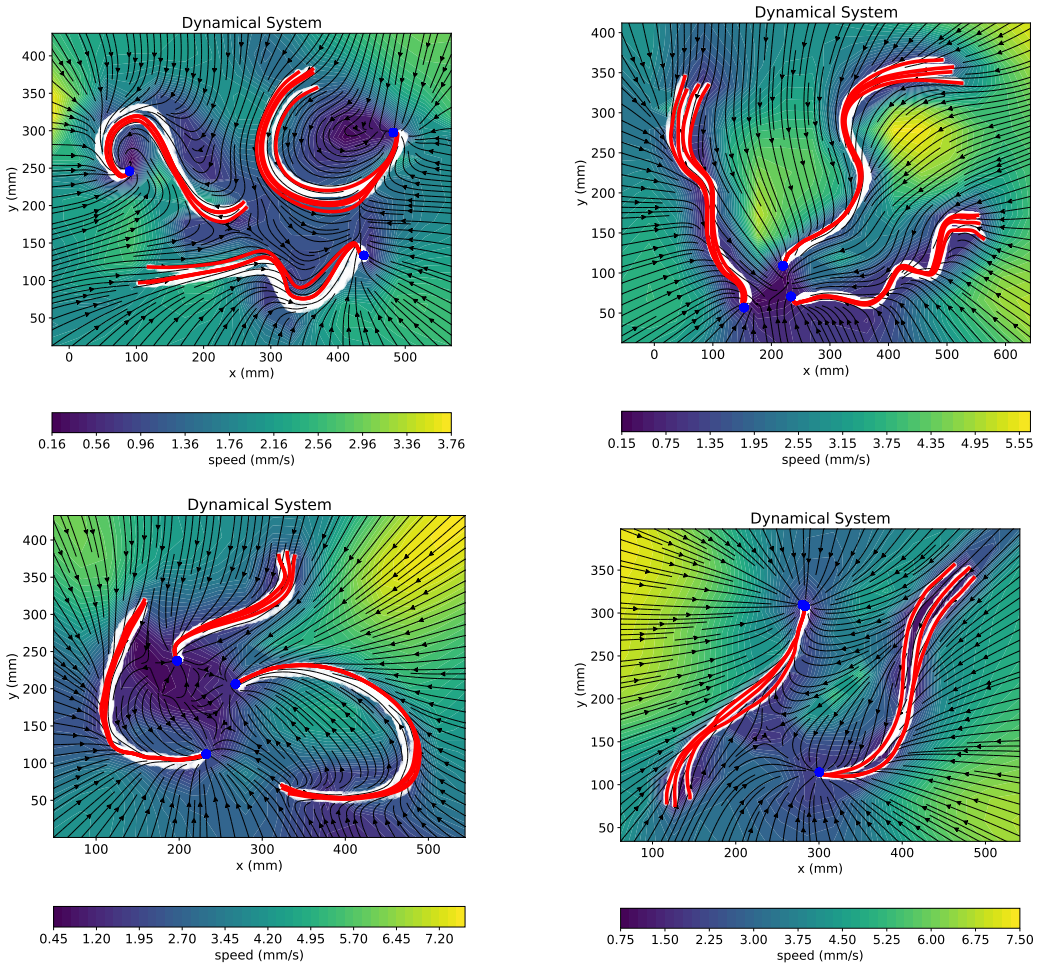


Figure 27: Example of learned trajectory and vector field for different demonstrations, using the developed system in combination with the Multivariate Gaussian Dynamic and the teacher forcing approach.

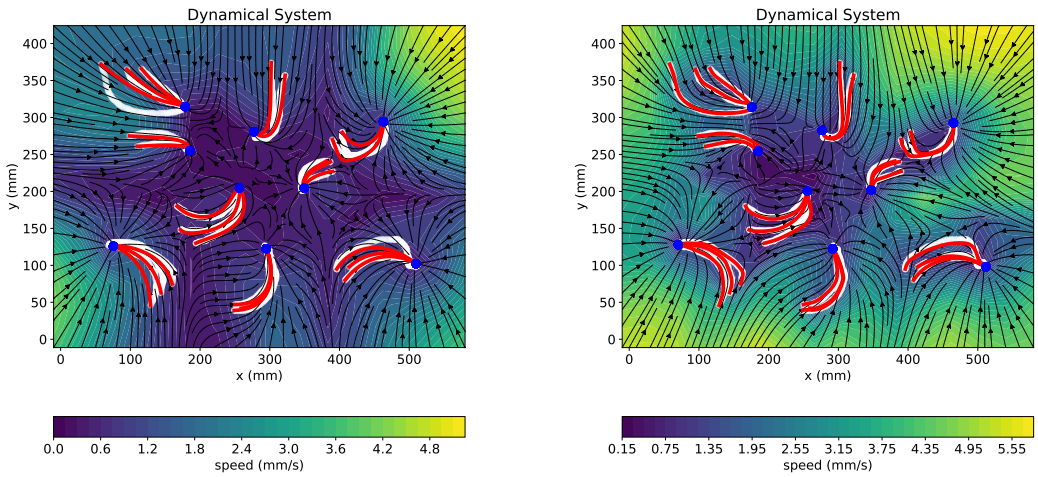


Figure 28: Experiment with larger number of distinct demonstrations. The first image is obtained when the mapping loss weight is still greater than 0; in the second, the mapping loss effect disappears.

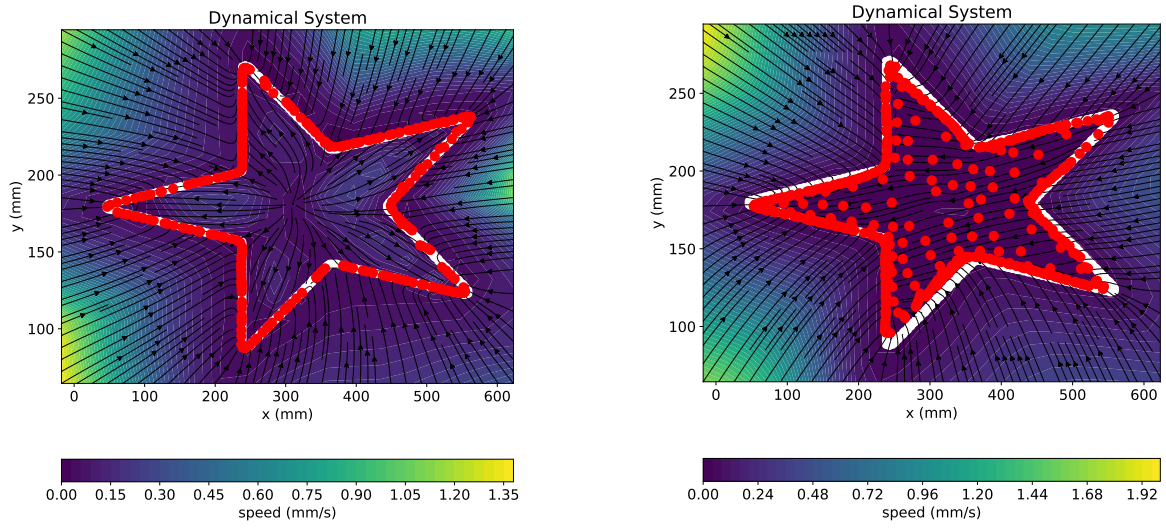
5.2 Continuous Case

5.2.1 Dynamics component

As described in the methodology section, different dynamics result in different vector fields. We recall that the continuous latent dynamic is partitioned into two sub-dynamics. The first must remain the same across all cases and is required to describe a limit cycle with angular velocity equal to 0, while the second can vary, provided it represents a single-attractor dynamic.
We also recall the two options used for the linear dynamic:

$$\dot{y}_{[:2]}(t) = -\alpha y_{[:2]}(t) \quad \text{and} \quad \dot{y}_{[:2]}(t) = -\alpha(y_{g_0} - y_{[:2]}(t))^2$$

where y_{g_0} represents the average of the latent points that define the shape.
The key difference lies in the selection of the attractors for the linear sub-dynamic. If we choose the zero vector $\mathbf{0}$ as the attractor, the resulting equilibrium points lie only on the contour of the shape. This leads to a distinct velocity profile within the shape's interior.
In the first implementation, only the points along the curve act as attractors. In contrast, the second approach also identifies internal points as attractors (with the methodology described in the Evaluation chapter 4).
Examples of the two vector fields are shown in Figure 29.



(a) Linear component equilibrium at $\bar{0}$
$\dot{y}_{[:2]}(t) = -\alpha y_{[:2]}(t)$

(b) linear equilibrium at y_{g_0}
$\dot{y}_{[:2]}(t) = -\alpha(y_{g_0} - y_{[:2]}(t))$

Figure 29: effect of the linear dynamics component of the continuous curve of attraction

As is often the case in engineering, there is no universally best solution, but only the most suitable one for a given application. When the objective is to reach the contour of an object, the most effective results are achieved using dynamics with an equilibrium

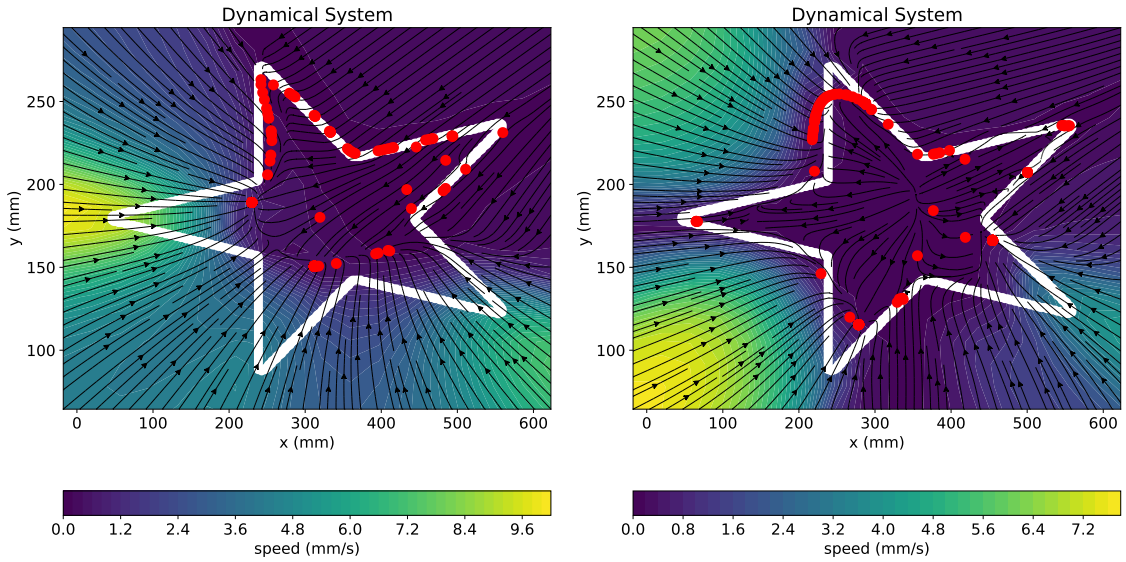


Figure 30: Training results without the hard negative sampling strategy. The first plot shows the model output without the orientation loss, while the second includes the orientation loss.

at the zero vector, expressed as $\dot{y}(t) = -y(t)$. Conversely, if the goal is to nullify the velocity field within the shape, the optimal approach is to select the average point of the shape as the sole equilibrium.
We now continue our analysis by focusing on the linear dynamics with equilibrium at $\mathbf{0}$, as this approach aligns more closely with our intended objective.

5.2.2 Contrastive norm results

The introduction of the contrastive norm is essential for enabling the model to learn the target shape. Without this term, the model fails to capture a meaningful latent representation of multiple distinct attractor points, often collapsing into a trivial solution characterized by a single-point attractor. Even more significant is the effect of combining the hard negative sampling strategy with the orientation loss.
Consider the scenario in which the model is trained with the contrastive loss but **without hard negative samples**. In this case, the learned attractor may still reflect aspects of the underlying latent dynamics — for example, by forming a continuous circular attractor. For such simple shapes, the model manages to create a repulsive region within the figure. However, its ability to generalize to more complex shapes is severely limited, as shown in the first plot of Fig. 30. Depending on the model's weight initialization, it may even fail to reconstruct simpler shapes entirely.
The addition of the orientation loss improves the model's capacity to generate a repulsive field inside the object's contour, even in the case of more complex shapes. Nonetheless, the reconstruction of the full target shape remains incomplete. Figure 30 illustrates the influence of the contrastive loss—with and without the orientation loss—when negative samples are restricted to points from the workspace.

5.2.3 Orientation error and loss comparison

We evaluated the performance of several loss functions in the context of orientation learning. Specifically, we report the $\updownarrow_1$ norm of the latent orientation error, alongside additional metrics that assess the system's ability to reconstruct the target shape.
This experiment uses the star-shaped figure as the reference dataset, chosen due to the inherent complexity it poses for the model. Each variation of the target shape introduces unique challenges, and the variance across datasets is relatively high.
To ensure a fair comparison, each training configuration was run five times with different random seeds. We then report the average values of the evaluation metrics along with their standard deviations. As shown in Table 2, the best performance is achieved using the MSE loss in combination with the trigonometric mapping $z(\theta) = [\sin(\theta), \cos(\theta)]^\top$. This mapping also proves beneficial when used in conjunction with the contrastive loss function.

Table 2: Effect of different loss on the orientation mapping

Loss Type	# Orientation Error	# Chamfer Distance	# Spurious Attractors
Without orientation mapping	1514.73 ± 89.60	32.71 ± 6.74	4.85%
Contrastive Loss on θ	8.12 ± 3.94	168.27 ± 12.27	17.18%
Contrastive Loss on $Z(\theta)$	5.75 ± 1.18	37.30 ± 2.41	4.42%
MSE Loss on θ	7.64 ± 1.34	11.07 ± 2.40	1.41%
MSE Loss on $Z(\theta)$	6.07 ± 1.08	8.14 ± 0.05	1.53%

A quantitative analysis of the generated plots reveals that the contrastive loss typically enables accurate reconstruction of only random segments of the target shape.

This behavior suggests that the loss margin is being leveraged to compensate for residual errors during the optimization process. We experimented with various margin values, selected through standard hyperparameter optimization using Optuna.

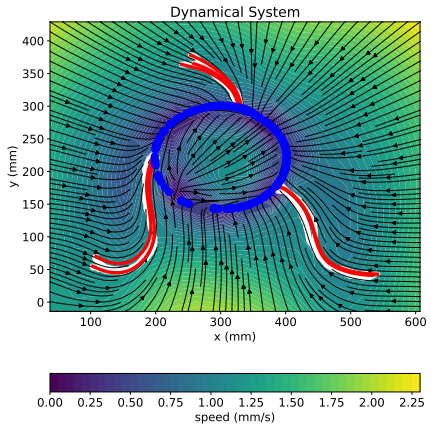


Figure 31: Integration of the continuous curve with 3 different demonstrations

5.2.4 Shapes reconstruction with LfD

Although the continuous attractor system is capable of learning complex shapes, integrating this framework with an imitation loss term, defined as $\mathcal{L}_{\text{imit}} = \text{MSE}(y(t)_{\text{demo}}, \hat{y}(t))$, proves challenging for non-trivial geometries. In our research, we combined Learning from Demonstration (LfD) with the continuous shape system, where the attraction trajectory is modeled as a circle passing through the N goal points. Examples of the resulting behavior are shown in Fig.31.

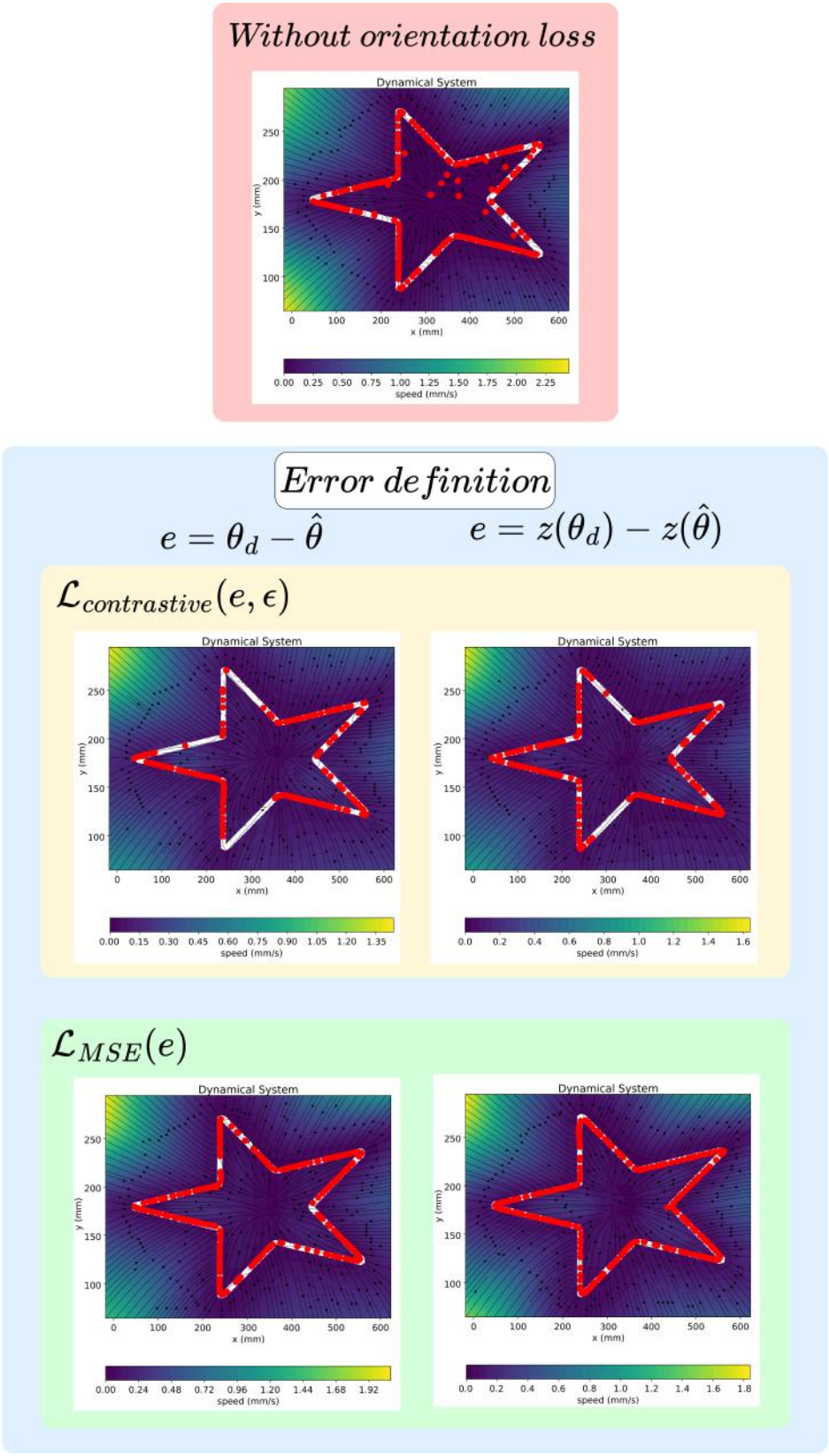


Figure 32: Comparison of the different effects of the orientation learning loss, for different formulations.

It is important to note that our models were only successful in reproducing relatively simple geometric shapes; they consistently failed to capture more complex forms when we want to impose a bias on the vector field.

5.3 Real-world experiments

Let's now see how the proposed approach works with real trajectories collected with the robotic manipulator. In this section we review the tool utilize for a grasping task with multiple target points. The experiment have been carry on first on simulation, and then on the real hardware. The task that we select for our experiment are a grasping task in which the robot should lift a box with two handle (symmetrically distributed on the object).
The low-level controller continuously maps the desired state x_d and its derivative $\dot{x}_d$, provided by the neural models, to the actuation torque vector u. To compute the corresponding input vector, we use the real-time inverse kinematics library TRACK-IK [34]. The models output the desired velocity $\dot{x}_d$, which is integrated using an Euler scheme to obtain the desired state x_d. These two quantities are then fed to the controller for torque computation.
Both solutions for the discrete attractor case were evaluated on 3D trajectories using demonstrations recorded with the Iiwa14 robot via kinesthetic teaching. In both scenarios, the attraction points were successfully reached. However, we observed a degradation in performance when a large number of demonstrations were used. We attribute this to the potential overlap of trajectories, which may introduce conflicting contrastive signals during model training (see Fig. 34). Experiments using a smaller set of demonstrations yielded higher accuracy compared to the over-demonstrated task.
While Condor is capable of controlling the end-effector's position, Puma also accounts for orientation. For this reason, the robotic experiments were conducted using the Puma framework. To evaluate Condor's effectiveness in a real-world scenario, we reconstructed the trajectories while disregarding the orientation component.
Finally, we present the evolution of the 6D trajectory obtained from Puma, where both position and orientation converge over time toward the desired state (Fig. 35).

5.3.1 Software set up

The robotics experiments were conducted using a complete simulation suite, involving ROS1-Noetic (Robot Operating System, [35]) for communication and Docker [36] to virtualize the robot environments.
Docker is a platform that automates the deployment of applications within lightweight, portable containers. These containers bundle the software along with its dependencies, ensuring consistent performance across different computing environments. In robotics, Docker addresses the challenge of managing complex software stacks, which often

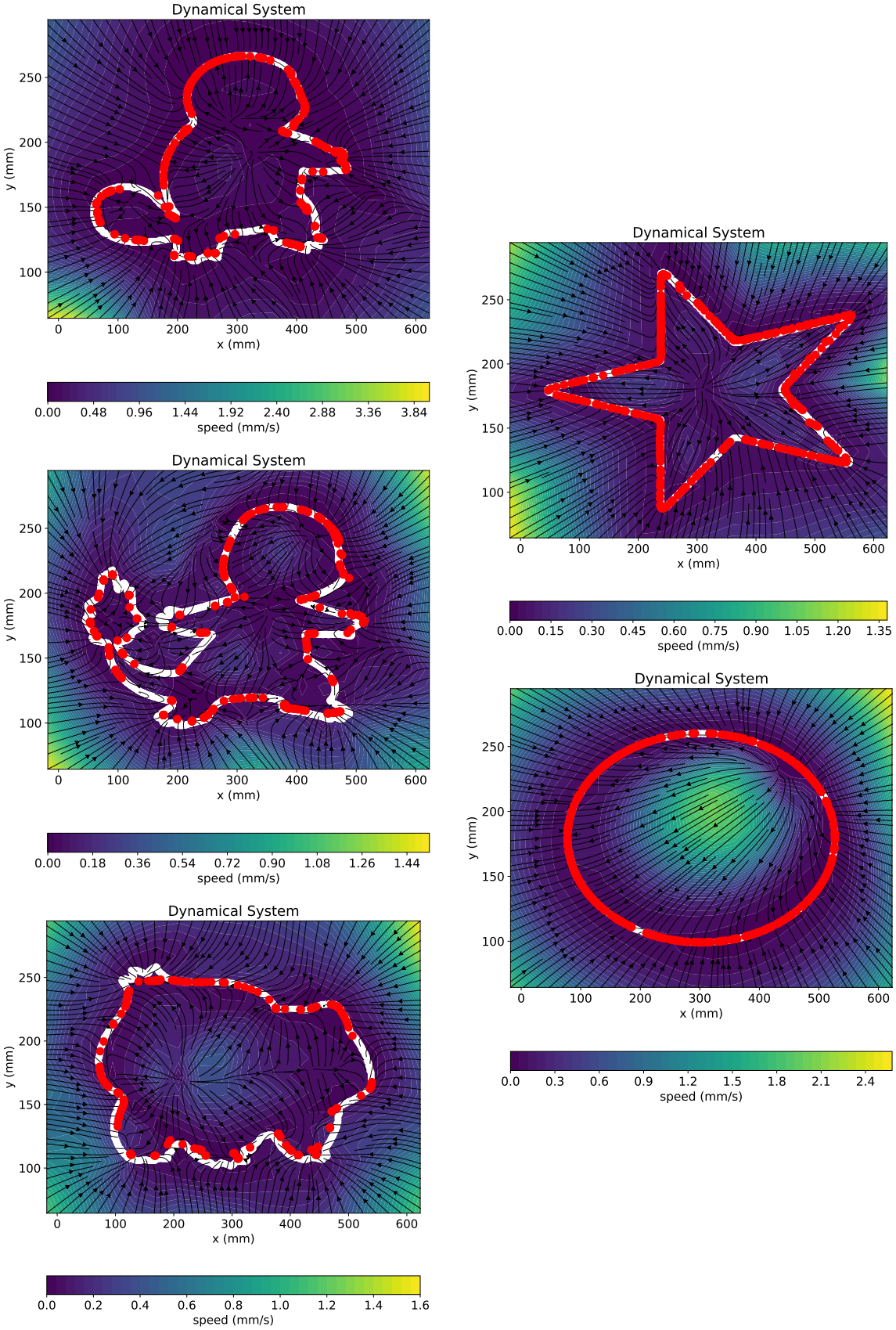


Figure 33: Comparison of learned trajectories and vector fields using different reference shapes.

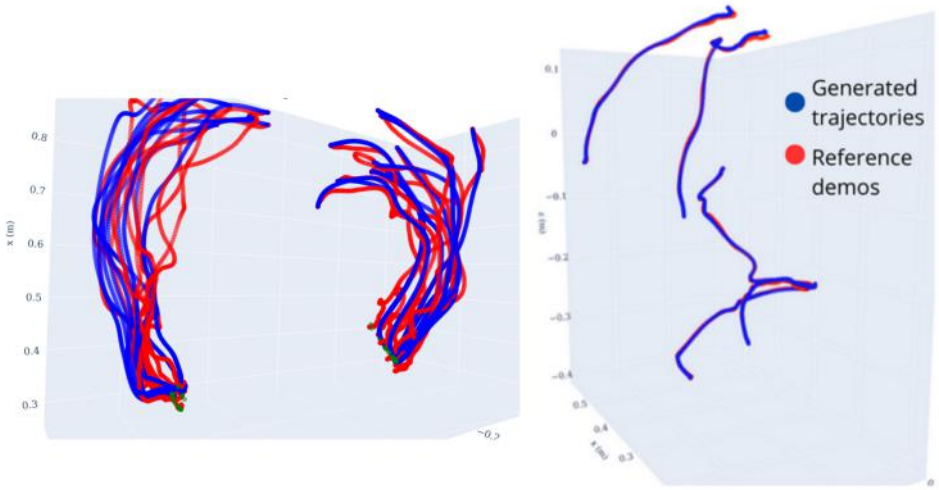


Figure 34: Trajectories generated with the multivariate extension of Condor (first image) and with the extension of Puma (second image), in a 3D space

include middleware, control algorithms, perception modules, and simulation tools.
To expose all necessary hardware interfaces to the Docker containers, a Bash script is employed. This script provides access to critical system components such as networking, the GPU, and the X-server—essential for interacting with graphical user interfaces (GUIs) in a Linux environment.
The system also relies on the IIWA driver and related packages, along with the proprietary Fast Research Interface (FRI), which enables real-time communication with the KUKA robotic arm.
Meanwhile, the Robot Operating System (ROS) acts as the middleware, managing communication between distributed software components called nodes. These nodes interact asynchronously via a publish-subscribe mechanism, an architecture particularly well-suited for real-time robotics applications. ROS also offers key tools such as RViz for 3D visualization and Gazebo for physical simulation, enabling efficient development and testing in virtual environments.
In our setup, simulation was carried out using the Gazebo simulator (see Fig. 36b), while demonstrations were collected using the physical robots.

5.3.2 Hardware

As previously mentioned, we employed a 7-DoF robotic manipulator—the KUKA iiwa 14. Communication with the robot's joints was established through the Fast Research Interface (FRI), which enables real-time control and data exchange.
The manipulator was equipped with an anthropomorphic soft hand developed by QB-Robotics (Fig. 36a). This soft hand features a native ROS interface, allowing seamless integration and control within our robotic framework. Its actuation mechanism relies on soft tendons driven by a single motor, classifying it as an underactuated

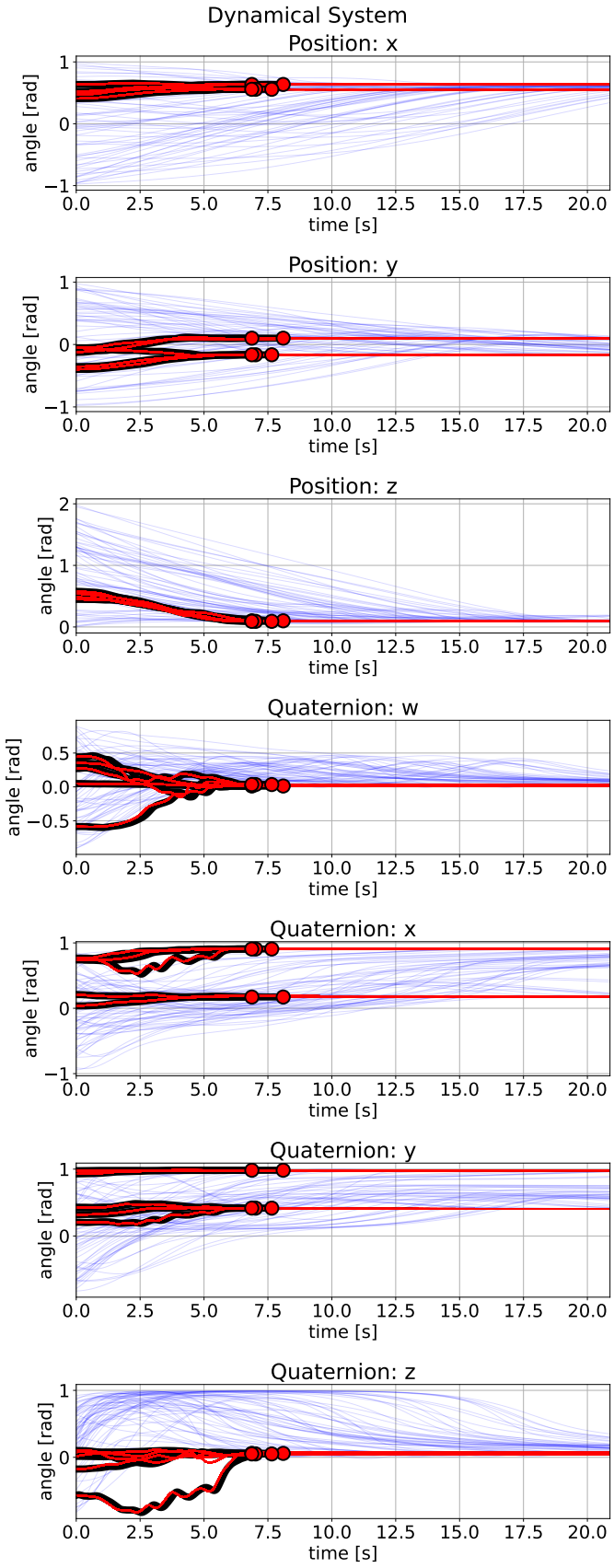
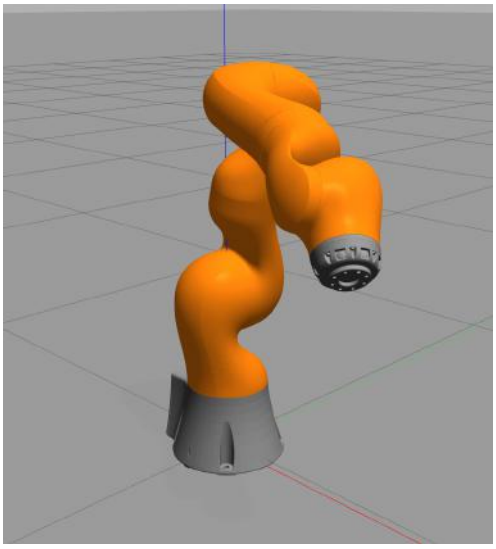


Figure 35: Evolution of the component of the state with Puma (2 attractors)

(a) The anthropomorphic soft hand provided by QB Robotics.



(b) Simulated KUKA arm in Gazebo.

Figure 36: Overview of the robotic hardware and simulated setup.

system capable of efficiently replicating human-like grasping behavior.
The experimental workspace was also equipped with multiple cameras from the OptiTrack motion capture system. OptiTrack is a high-precision tracking solution widely used in robotics and biomechanics for real-time 3D motion capture. It employs an array of infrared cameras to detect reflective markers affixed to rigid bodies, enabling reconstruction of their position and orientation. With low-latency data streaming and robust integration capabilities, OptiTrack is well-suited for real-time control and feedback. In our setup, it was used to measure the displacement of the target object relative to a reference position.

5.4 Grasping task

To validate our solution, we recorded a set of 10 demonstrations (5 for each attractor), where the robot reaches a grasping configuration. The endpoints of these trajectories were designated as attractor points. The robot successfully reached each of the two grasping targets by following the reference poses generated by the neural model.
The control loop operates as a finite state machine, where a set of Boolean variables is updated based on the robot's current position. Key poses stored during execution include the home pose and the goal pose, the latter corresponding to the selected attractor. Once the goal pose is reached, the state machine triggers the gripper to close, completing the grasping task. Finally, the robot returns to the home configuration
We observed that in certain configurations, the robot may encounter a singularity, in which the low-level controller—based on an online inverse kinematics solver—is unable to accurately follow the trajectory generated by the neural model.
Despite this limitation, the model demonstrated robustness to perturbations, as shown

in the first row of Fig. 38, proving the efficiency of the stability-based approach.

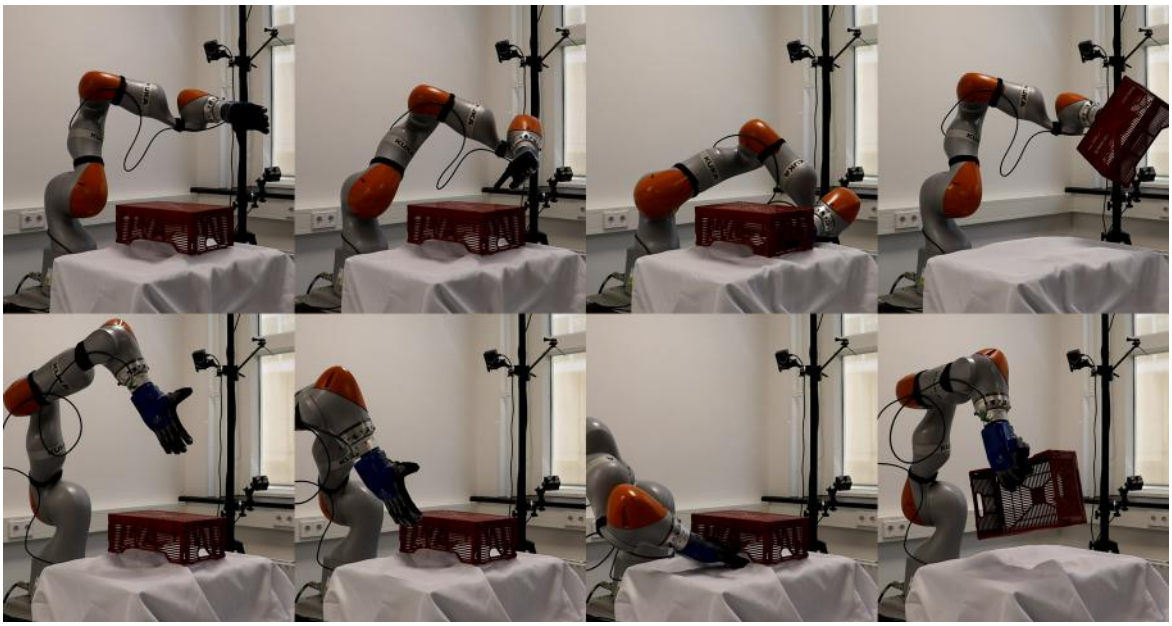


Figure 37: 2 execution of the grasping task starting from different initial condition. The robot reach different grasping point, conditioning the starting position

5.4.1 Target in motion

The presented framework is designed to learn how to reach a fixed target point within the workspace. However, modern robotic systems must also be capable of adapting to small environmental changes. To address this limitation, we integrated a motion tracking system to monitor the real-time position of the target endpoints and compute their displacement relative to the original attractor points. Markers were placed on the corners of the target object (a box), allowing the OptiTrack system to continuously track its position.

Once the neural model outputs the desired end-effector velocity to reach the target, we integrate this velocity using an Euler scheme to obtain the reference position for the inverse kinematics controller. Before sending the desired state x_d to the controller, we apply the position offset measured by the OptiTrack system. This adjustment ensures robust goal tracking even in the presence of target perturbations.

In the first row of Fig. 38, an operator physically displaces the robot from its starting configuration. This scenario is used to evaluate the model's performance when initialized from random positions within the workspace. The manipulator continues to move toward its goal even as the attractor point (the box) is moved around the workspace, as shown in the second row of Fig. 38, thereby demonstrating the system's ability to handle target displacement.

Figure 38: Grasping task with a perturbed initial position (first row) and a moving object (second row)

6 Future Works

The proposed approach demonstrates strong performance across a wide range of tasks and demonstrations. However, it is important to acknowledge several limitations and outline directions for future research to address these gaps in the methodology.

The most evident limitation lies in the lack of orientation-level control within the multi-Condor framework. Although we experimented with various distance metrics, it appears that the ability to map task-space trajectories onto a manifold is intrinsically tied to the formulation of the latent system dynamics. For instance, Puma utilizes the derivative of the encoder to define the latent dynamics, effectively addressing the manifold mapping problem. However, this comes at the cost of model interpretability, as it eliminates the use of hard-coded system dynamics discussed in earlier chapters.

We also explored the representation of orientation as a Euclidean vector by applying the logarithmic map of $\mathcal{SO}(3)$, thus operating within the tangent space of the manifold. Despite this, our results indicated that the multi-Condor model was unable to accurately learn orientation information.

Looking ahead, it is worth exploring the integration of more complex dynamical systems into our framework. One promising direction is the use of harmonic oscillator dynamics, which can offer a deeper physical interpretation of the underlying behavior of the system. In particular, as discussed in [37], harmonic oscillators have been shown to effectively model attraction phenomena in latent spaces, and their integration may help structure the latent dynamics in a more interpretable and controllable way.

We also considered the potential of Kuramoto models, which consist of networks

of coupled oscillators. Their synchronization behavior could be useful for capturing more complex latent interactions, especially in continuous-attraction settings. In such scenarios, we currently rely on only the first two equations of latent dynamics to generate the attraction curve, which may under-utilize the capacity of the latent space. Synchronization models like Kuramoto could provide richer temporal or phase-based coordination across latent dimensions, potentially leading to more expressive models. That said, the accurate replication of attraction curves in three-dimensional spaces remains a challenging problem. The complexity arises from both the geometric nature of 3D space and the limitations of the current latent representations. Future research should further investigate how advanced dynamical models can support not just 2D planar convergence, but more structured and controllable 3D attraction behaviors as well. Moreover, move to a coupled oscillator system can recall what have been done in [38], where a mechanism for control the attractors switching have been designed.

Another possible area of future improvement is a selection scheme that allow the model to select the best motion attractor, as for the moment the learned vector field is static. Introduction of tag variable into the state can partially solve the problem, as explored also in the multi-primitive version of Condor [17].
Finally, we propose a potentially paradigm-shifting enhancement: the integration of Geometric Algebra (GA), also known as Clifford Algebra, into the framework. GA offers a powerful mathematical language that extends classical vector algebra by incorporating multivectors, which can represent scalars, vectors, areas, volumes, and higher-order geometric entities. Geometric calculus, a subfield of GA, provides a way to perform differential and integral calculus in this expanded space. Applying GA to describe latent dynamics could unify the diverse components of our framework, offering both lower symbolic complexity and enhanced geometric interpretability. Examples of GA's advantages in robotic motion representation can be found in [39], and we believe its application here could significantly enhance the expressiveness and clarity of our models.

7 Conclusions

In this thesis, we investigate the integration of multi-stable dynamics as a reference model within the Dynamic Movement Primitives (DMP) framework. Our primary focus is on point-to-point trajectory generation, using human demonstrations as reference motions.

A core requirement of our approach is stability. Achieving stable behavior while preserving the expressiveness of neural models presents a considerable challenge. To address this, we introduce a latent space representation within a neural network, shaped by a combination of loss components: stability loss, imitation loss, and mapping loss. Our method leverages a multivariate Gaussian dynamical system in the latent space, complemented by carefully designed loss functions that impose priors to guide the latent representations toward meaningful and controllable dynamics. Inspired by the Teacher Forcing paradigm, our training strategy ensures the network learns consistent attractor behaviors. We assess the effectiveness of our framework through both quantitative metrics and qualitative comparisons with established methods. The results demonstrate that our approach produces smoother trajectories across the robot's workspace, maintaining accuracy even as the number of attractor points increases.

The experimental validation included both simulated and real-world testing on a KUKA iiwa14 robotic arm, equipped with a soft anthropomorphic hand. The framework demonstrated robustness in reaching multiple grasping targets, successfully adapting to perturbations in the robot's initial configuration and even tracking displacements of moving targets via integration with a motion capture system. The neural network outputs desired end-effector velocities, which serve as references for a low-level controller. These are then translated into joint-space commands via an inverse kinematics solver. Our approach has promising applications in industrial environments, enabling untrained human operators to provide intuitive demonstrations. The ability to encode multiple trajectories within a single model is particularly valuable for tasks such as grasping symmetric objects or generalizing grasping strategies across varying positions—for instance, grasping a mug by the handle or from the top.

Extending beyond discrete goal representations, we introduce the concept of continuous attractor curves, where every point along a curve acts as a potential goal state. To model this, we map the attraction curve onto a unit circle within the latent space of the neural model. A dedicated set of loss functions has been developed to tailor the latent representation of the curve. By partitioning the latent dynamics into subcomponents, we enable the model to represent both the contour and the enclosed area as attractors in the continuous case. Our key contribution for this task lies in the introduction

of a contrastive norm loss, combined with a hard negative sampling strategy, which enhances the expressiveness and discriminative power of the latent space. Additionally, an orientation loss is introduced to ensure an even distribution of the attraction region along the contour.

With this work, we aim to provide a comprehensive overview of strategies for learning from multiple demonstrations simultaneously. Our approach contributes to the development of more adaptive and expressive robotic systems, paving the way for the robots of tomorrow. We hope this research facilitates the integration of advanced artificial intelligence techniques within the industrial robotics community, fostering more intuitive, flexible, and human-centered automation solutions.

References

[1] Michael Hagenow, Dimosthenis Kontogiorgos, Yanwei Wang, and Julie Shah. Versatile demonstration interface: Toward more flexible robot demonstration collection, 2025.

[2] Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress, 2020.

[3] A.J. Ijspeert, Jun Nakanishi, and Stefan Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. volume 2, pages 1398 – 1403, 02 2002.

[4] A. Ijspeert, J. Nakanishi, P Pastor, H. Hoffmann, and S. Schaal. Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Computation*, (25):328–373, 2013. clmc.

[5] Ashwini Shukla and Aude Billard. Augmented-svm: Automatic space partitioning for combining multiple non-linear dynamics. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[6] Ying Zhang, Miao Li, and Chenguang Yang. Robot learning system based on dynamic movement primitives and neural network. *Neurocomputing*, 451:205–214, 2021.

[7] Affan Pervez, Yuecheng Mao, and Dongheui Lee. Learning deep movement primitives using convolutional neural networks. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 191–197, 2017.

[8] Andreas Sochopoulos, Michael Gienger, and Sethu Vijayakumar. Learning deep dynamical systems using stable neural odes, 2024.

[9] Sebastian Bitzer and Sethu Vijayakumar. Latent spaces for dynamic movement primitives. In *2009 9th IEEE-RAS International Conference on Humanoid Robots*, pages 574–581, 2009.

[10] Nutan Chen, Justin Bayer, Sebastian Urban, and Patrick van der Smagt. Efficient movement representation by embedding dynamic movement primitives in deep autoencoders. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, page 434–440. IEEE Press, 2015.

[11] Jože M Rožanec and Bojan Nemec. Neural dynamic movement primitives – a survey, 2022.

[12] S. Mohammad Khansari-Zadeh and Aude Billard. Imitation learning of globally stable non-linear point-to-point robot motions using nonlinear programming. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2676–2683, 2010.

[13] S. Mohammad Khansari-Zadeh and Aude Billard. Learning stable nonlinear dynamical systems with gaussian mixture models. *IEEE Transactions on Robotics*, 27(5):943–957, 2011.

[14] Yanwei Wang, Nadia Figueroa, Shen Li, Ankit Shah, and Julie Shah. Temporal logic imitation: Learning plan-satisficing motion policies from demonstrations, 2022.

[15] Hiroyasu Tsukamoto and Soon-Jo Chung. Imitation learning for robust and safe online motion planning: A contraction theory approach. 2021.

[16] Sumeet Singh, Spencer M. Richards, Vikas Sindhwani, Jean-Jacques E. Slotine, and Marco Pavone. Learning stabilizable nonlinear dynamics with contraction-based regularization, 2019.

[17] Rodrigo Pérez-Dattari and Jens Kober. Stable motion primitives via imitation and contrastive learning, 2023.

[18] Rodrigo Pérez-Dattari, Cosimo Della Santina, and Jens Kober. Puma: Deep metric imitation learning for stable motion primitives, 2024.

[19] Muhammad Asif Rana, Anqi Li, Dieter Fox, Byron Boots, Fabio Ramos, and Nathan Ratliff. Euclideanizing flows: Diffeomorphic reduction for learning stable dynamical systems, 2020.

[20] Jiechao Zhang, Hadi Beik-Mohammadi, and Leonel Rozo. Learning riemannian stable dynamical systems via diffeomorphisms, 2022.

[21] S. Mohammad Khansari-Zadeh and Aude Billard. Learning stable nonlinear dynamical systems with gaussian mixture models. *IEEE Transactions on Robotics*, 27(5):943–957, 2011.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[23] Xin Jin and Jiawei Han. *K-Means Clustering*, pages 563–564. Springer US, Boston, MA, 2010.

[24] P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[25] Paul Dierckx. *Curve and Surface Fitting with Splines*. Oxford University Press, 03 1993.

[26] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[27] Florian Hess, Zahra Monfared, Manuel Brenner, and Daniel Durstewitz. Generalized teacher forcing for learning chaotic dynamics, 2023.

[28] Hongyun Zhang and Jin Liu. Direction estimation of aerial image object based on neural network. *Remote Sensing*, 14(15), 2022.

[29] Jie Sun, Wengang Zhou, and Houqiang Li. Orientation estimation network. In Yao Zhao, Xiangwei Kong, and David Taubman, editors, *Image and Graphics*, pages 151–162, Cham, 2017. Springer International Publishing.

[30] Hiroaki Sakoe. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26:159–165, 1978.

[31] Thomas Eiter and Heikki Mannila. Computing discrete frechet distance. 05 1994.

[32] Ainesh Bakshi, Piotr Indyk, Rajesh Jayaram, Sandeep Silwal, and Erik Waingarten. A near-linear time algorithm for the chamfer distance, 2023.

[33] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. *CoRR*, abs/1605.08803, 2016.

[34] Patrick Beeson and Barrett Ames. Trac-ik: An open-source library for improved solving of generic inverse kinematics. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 928–935, 2015.

[35] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66), May 2022.

[36] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[37] Maximilian Stölzle and Cosimo Della Santina. Input-to-state stable coupled oscillator networks for closed-form model-based control in latent space, 2024.

[38] Katsuma Inoue, Kohei Nakajima, and Yasuo Kuniyoshi. Designing spontaneous behavioral switching via chaotic itinerancy. *Science Advances*, 6(46), November 2020.

[39] T. Löw and S. Calinon. Geometric algebra for optimal control with applications in manipulation tasks. *IEEE Transactions on Robotics (T-RO)*, 39(5):3586–3600, 2023.

Acknowledgement

20 Maggio 2025
Questa parte della tesi e' l'unica che ho scritto piu' volte. E' anche l'unica parte che non vedevo l'ora di scrivere. Ci pensavo spesso, nei momenti no. Era una occasione per ringraziare tanta, tanta gente. Pensare a tutti voi mi faceva star meglio.

Per iniziare, ringrazio i miei genitori, e mio fratello, che mi sono stati accanto durante questi anni.

Questa tesi mi ha preso un anno intero della mia vita, questa laurea ne ha presi 4. I momenti grigi sono spesso stata la quotidianita', e quelli neri non sono stati nemmeno troppo rari. Se pero' oggi state leggendo questo, e' perche' devo tutto a certe persone, e con questa pagina vorrei ringraziarli.

Partiamo da una canzone. Nella prima stesura (mentale) di questo testo avevo deciso che ne avrei inserite molte. Non lo faro', ma una deve restare: 2050, degli Zen Circus.

> [...]
> Scisso gli atomi di una conchiglia
> Vinto la morte, perso la meraviglia
> Strappato foreste come fili d'erba
> Abbiamo dato un nome ad ogni stella
> Fatto l'amore senza capirne nulla
> Condannata la pace ad essere anche guerra
> Cambiato il corso dell'acqua corrente
> [...]
> Quando lui ti chiederà
> Che cosa ha fatto la gente
> Digli abbiamo fatto tutto
> Non abbiamo fatto niente.

Il cantante racconta il suo passato a suo figlio, dalla prospettiva del futura. Futuro e' anche stata la parola chiave di questi anni, sembra che ogni cosa dovesse avere inciso un qualcosa che alludesse al futuro, a caratteri cubitali.

Ogni volta che sento questa canzone non posso che pensare a tutti voi, gente del DIAG. Pazzi colleghi che sarebbero capaci di tutto, se volessero. Che hanno davvero dato un nome ad ogni stella. Ogni progetto, ogni lezione, ogni serata, ogni canzone. Abbiamo davvero fatto tutto? Con gente del Diag non intendo solo i colleghi in senso lato,

attorno a noi si e' creato un ecosistema di persone straordinarie. Alcuni del DIAG non erano gente del DIAG, mentre persone che nel dipartimento ci stavano effettivamente non sono mai state persone del DIAG.

Ad un certo punto del percorso credo che un po' tutti noi abbiamo "perso la meraviglia". Vuoi perche' il mondo ci ha schiaffeggiato, vuoi perche col tempo alla meraviglia ci si abitua e l'assurdo diventa ordinario.
E' passato piu di un anno da quando ho lasciato Roma e da allora non ho piu trovato un gruppo di persone cosi in sintonia, cosi veloci e creative. Da quando sono andato via, la mia vita ha assunto una nota di ordinario totalmente diversa, e nulla penso sara' mai paragonabile a quello che abbiamo fatto in quegli anni.
Noi abbiamo fatto tutto. Abbiamo dato tutto. Per un po' ci e' sembrato non ci dovesse rimanere niente. Non valeva a nulla tutta la fatica fatta, nemmeno l'impegno che ci abbiamo messo.

Sara' la stanchezza che mi rende malinconico, o forse questo e' solo un lato del mio carattere che cerco di non far emergere. Di certo non emergeva con voi, non c'e mai stato spazio per la malinconia. Nemmeno nei momenti piu tristi.
Oggi, con questo pezzo di carta stampata, voglio far si che rimanga un ennesimo ricordo per le persone che piu' di tutte mi hanno spinto avanti in questi anni.

Per ogni volta che ho brindato con voi, che ho cantato e che ho pianto con voi. Per ogni momento in cui avete creduto in me piu' di quanto io credessi in me stesso. Mi avete dato prospettive diverse, sincere e critiche. Mai dure, ma non per questo non esigenti. Per ogni volta che qualcuno mi ha preso per le spalle scuotendomi, ripetendomi un "Gabrie', sei forte, ce la fai", per tutte le volte siamo stati in disaccordo e abbiamo fatto le 4 a discutere sul giusto accento di una parola e per tutte quelle sere sui banchi in cui ci siamo sentiti meno soli. Per i discorsi tra i fumi dell'alcol da cui nascevano idee e risate. Con voi ho passato persino dei Natali, sentendomi accolto e sempre ben voluto.
Vi voglio ringraziare perche' mi avete dato un letto quando ne avevo bisogno, perche' siete stati con me nei momenti piu soli, perche' quando vi ci mettevate avevate il potere di rendere possibile l'impossibile.

In Olanda poi, quanto siete serviti. Senza le vostre chiamate infinite non sarei sopravvissuto. Non avrei nemmeno saputo quando era il momento di fermarmi a respirare, per capire cosa fosse giusto nella mia vita e cosa vi fosse di sbagliato. La canzone del DIAG e' un pezzo che tuttora mi scalda il cuore e mi da serenita'.
Mi avete inspirato, supportato, tenuto assieme. Siete sempre riusciti a riaccendere una scintilla in me, che spesso divampava in una fragorosa risata.

Sembra che non sia mai esistito un gruppo cosi forte, cosi ricco di unicita'. Cosi' speciale. Ogni singola cosa che ho ragiunto negli ultimi 4 anni non sarebbe stata possibile senza uno di voi, bastava che ne mancasse uno, e sarebbe crollato tutto.

Mattia, Peppe, Diego, Antonio.
Scardino, Nick e Martina.
Marco, Francesca, Pasquale, Rossana.
Riccardo e Gerardo, Massimo, Michele.
Salvo.

Voglio dedicarvi questa tesi, perche' siete l'unica cosa che non avrei potuto trovare in nessun altro posto al mondo.

Ceci, con te e' un discorso a parte. Mi hai supportato piu di tutti, e sei rimasta con me in una strada fatta totalemnte di ostacoli. Ti voglio bene e ti sono grato per ogni minuto speso al telefono. Se non fossi stata qui, non avrei mai superato certi ostacoli, hai risolto la meta' dei miei problemi e mi hai aiutato con l'altra meta' ancora irrisolta. Ancora, ti voglio bene, grazie.

Altre persone hanno costellato la vita Romana. Flavio, Diana, Andrea. Anche voi siete stati parte di questo processo e non sapete quanto mi avete fatto crescere, professionalmente e umanamente.

Voglio anche ringraziare quel gruppo di persone di Palermo che mi e' stato dietro fin da bambino. Loro sono stati tra gli amici piu storici, e mi sono stati vicino nel rientro in Sicilia e non solo. Dario e Lorenzo, Rocco e Morale. Avete assistito a questa tesi e ai suoi sviluppi, condividendo l'entusiasmo per qualche grafico incomprensibile e accettando quello che realizzarlo ha comportato. Vi voglio bene, e sono felice di avervi avuto con me in questi anni.

Matteo, tu ti meriti una sezione tutta tua. Non so davvero cosa dirti, ma non penso ci sia qualcosa di importante che tu non sappia gia'. Se non fosse stato per te non so come sarei cresciuto, probabilmente ancora avrei difficolta' a parlare con le persone e me la vivrei dannatamente male. Se ai colleghi ho dedicato la tesi stessa, il nome del progetto e' tutto in tuo onore. SQUID. Animale che ti ha sempre caratterizzato. Credo che non esista persona al mondo che sappia leggermi meglio di te. Con te abbiamo condiviso sangue e sudore in bici, sporco e colori nell'arte, idee e dialoghi nella vita. Se mai qualcuno sara' fiero di quello che sono, sappi che e' merito tuo. Grazie.

Gabriele