A structured prediction approach for robot imitation learning

Journal Title
XX(X):1–19

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

A Structured Prediction Approach for Robot Imitation Learning

Anqing Duan[1], Iason Batzianoulis[2], Raffaello Camoriano[3], Lorenzo Rosasco[4, 5, 6], Daniele Pucci[7], and Aude Billard[2]

Abstract

We propose a structured prediction approach for robot imitation learning from demonstrations. Among various tools for robot imitation learning, supervised learning has been observed to have a prominent role. Structured prediction is a form of supervised learning that enables learning models to operate on output spaces with complex structures. Through the lens of structured prediction, we show how robots can learn to imitate trajectories belonging to not only Euclidean spaces but also Riemannian manifolds. Exploiting ideas from information theory, we propose a class of loss functions based on the f-divergence to measure the information loss between the demonstrated and reproduced probabilistic trajectories. Different types of f-divergence will result in different policies, which we call *imitation modes*. Furthermore, our approach enables the incorporation of spatial and temporal trajectory modulation, which is necessary for robots to be adaptive to the change in working conditions. We benchmark our algorithm against state-of-the-art methods in terms of trajectory reproduction and adaptation. The quantitative evaluation shows that our approach outperforms other algorithms regarding both accuracy and efficiency. We also report real-world experimental results on learning manifold trajectories in a polishing task with a KUKA LWR robot arm, illustrating the effectiveness of our algorithmic framework.

Keywords

Imitation learning, structured prediction, learning and adaptive systems, kernel methods, Riemannian manifolds

1 Introduction

The general notion of imitation is widely exploited in robotics as it can bring multiple benefits (Osa et al. 2018). For example, imitation learning has been proven to be an effective approach to facilitate the acquisition of motor skills for complex high-dimensional humanoid robots (Schaal 1999; Yang et al. 2018). Also, imitation learning can be employed for robots to achieve tasks whose rewards are intricate to manually specify, such as aerobatic maneuvers for helicopter flight (Abbeel et al. 2010) and dynamic flips and spins (Peng et al. 2018). Besides, by improving policies initialized by imitation learning, reinforcement learning can converge faster than optimizing a policy from scratch (Kober and Peters 2014; Cheng et al. 2018).

Briefly, there are two major paradigms for implementing imitation given expert demonstrations. The first is centered around the policy, where a policy is directly learned by applying a supervised learning algorithm to find a mapping from input states and context factors to output actions (Billard et al. 2008). The second is centered around the reward, where an unknown reward function is recovered through inverse reinforcement learning or inverse optimal control (Abbeel and Ng 2004; Ratliff et al. 2009).

In this paper, we focus on policy-centered imitation learning. Particularly, we consider the scenario where policy representation is instantiated with a trajectory-level abstraction. In this context, imitation learning is also known as programming by demonstration, where a learner's motion skills are usually acquired by penalizing deviation from the demonstrated trajectory.

Notably, movement primitives remain a central research topic in trajectory imitation with the goal of encoding motor skills from demonstrated trajectories for subsequent usage (Ravichandar et al. 2020). To this end, various supervised learning algorithms have been leveraged (Stulp and Sigaud 2015). More specifically, regression techniques, either parametric or non-parametric, constitute a significant contribution to the development of movement primitives. In the following, we briefly cover relevant works highlighting their strengths and limitations.

Dynamic Movement Primitives (DMP) is one of the pioneering imitation learning algorithms to mimic the expert's trajectory (Ijspeert et al. 2013). It has been gaining

[1]Robotics and Machine Intelligence Laboratory, The Hong Kong Polytechnic University, Hong Kong SAR, China
[2]Learning Algorithms and Systems Laboratory, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
[3]Visual And Multimodal Applied Learning Laboratory (VANDAL), Politecnico di Torino, Turin, Italy
[4]DIBRIS, Università degli Studi di Genova, Genoa, Italy
[5]Laboratory for Computational and Statistical Learning (IIT@MIT), Istituto Italiano di Tecnologia and Massachusetts Institute of Technology, Cambridge, MA, United States
[6]Machine Learning Genoa (MaLGa) Center, Università di Genova, Genoa, Italy
[7]Artificial and Mechanical Intelligence research line (AMI), Istituto Italiano di Tecnologia, Genoa, Italy

Corresponding author:
Raffaello Camoriano, Politecnico di Torino, C.so Francesco Ferrucci 112, 10141 Turin, Italy.
Email: raffaello.camoriano@polito.it

Table 1. Comparison of features with respect to state-of-the-art methods.

	Probabilistic	Trajectory modulation	Manifold output	Manifold modulation	Multiple imitation modes	Multi-dim input	Multiple output types
DMP (Ijspeert et al. 2013)	-	✓	-	-	-	-	-
ProMP (Paraschos et al. 2013)	✓	✓	-	-	-	-	-
LAT (Reiner et al. 2014)	✓	-	-	-	-	-	-
GMM (Zeestraten et al. 2017b)	✓	-	✓	-	-	✓	-
LGP (Schneider and Ertel 2010)	✓	-	-	-	-	-	-
TLGC (Ahmadzadeh and Chernova 2018)	-	-	✓	-	-	-	-
KMP (Huang et al. 2019)	✓	✓	-	-	-	✓	-
LPV-DS (Figueroa and Billard 2018)	-	✓	✓	✓	-	✓	-
Our Approach	✓	✓	✓	✓	✓	✓	✓

popularity, as evidenced by its large number of derivatives, such as sequenced DMP (Kulvicius et al. 2011), generalized DMP (Zhou and Asfour 2017), constrained DMP (Duan et al. 2018), etc. Recent advances in DMP such as Neural Dynamic Policies (NDPs) integrate deep learning to handle high-dimensional inputs like visual data (Bahl et al. 2020). The conception of DMP is based on the spring-damper dynamic system, whose acceleration profile is fitted with a set of manually defined basis functions to capture the shape of a demonstrated trajectory. DMP supports goal adaptation, yet it cannot handle via-point constraints and multiple demonstrations. To overcome the limitations, Probabilistic Movement Primitives (ProMP) was developed to learn a distribution over trajectories (Paraschos et al. 2013). Besides, ProMP can execute via-point trajectory adaptation, achieved by Gaussian conditioning.

In contrast to DMP and ProMP, motion imitation can also be realized from a non-parametric angle. For example, Kernelized Movement Primitives (KMP) leverages the kernel trick for movement representation (Huang et al. 2019). Due to the kernel formulation, it is straightforward for KMP to handle multi-dimensional inputs, which could be exploited in human-robot collaboration or task synergy retrieval (Zeestraten et al. 2017a). Other non-parametric methods built upon Gaussian processes, such as Local Gaussian process regression (LGP) (Schneider and Ertel 2010) and Gaussian process Models (GPM) (Arduengo et al. 2021), can learn the demonstrated trajectory as well.

Moreover, autonomous dynamical systems are also powerful tools for imitation learning. For example, Stable Estimator of Dynamical Systems (SEDS) (Khansari-Zadeh and Billard 2011) or Linear Parameter-Varying Dynamical System (LPV-DS) (Figueroa and Billard 2018) drops explicit time dependency and can ensure global stability while being robust to external disturbances. Dynamical systems can be extended to control forces at the contact level (Amanhoud et al. 2019). Learning approaches can be used to model both the motion and force profile (Khoramshahi et al. 2020) and adapt to a prescribed surface.

Despite the aforementioned advancements, so far only the imitation of trajectories in Euclidean spaces has been investigated, leaving the issue of learning trajectories with manifold constraints relatively under-explored. Arguably, many imitation objectives in robotics involve the analysis of geometry-structured training data, such as rotation matrices (Traversaro et al. 2016), stiffness ellipsoids (Ajoudani et al. 2018), etc. More importantly, it can be safety-critical to avoid breaking manifold-imposed constraints in some applications (Ahmadzadeh and Chernova 2018; Duan et al. 2022). Driven by theoretical questions and practical gains, it is thus crucial to develop imitation learning algorithms that are applicable to Riemannian manifolds.

In this paper, we present a persistent algorithmic framework for probabilistic imitation learning. The key novelty in our approach is to adopt a structured prediction formulation for robot imitation learning. Structured prediction enables complex outputs and can deal with trajectories on manifolds. Our proposed approach can handle the imitation of trajectories lying in either Euclidean space or a manifold, whilst also preserving the essential functionalities for movement primitives. Thanks to the inherent kernel method, our approach admits a non-parametric formalism and can learn trajectories driven by multi-dimensional inputs.

When carrying out probabilistic trajectory imitation, it is necessary to specify a suitable loss function that measures the discrepancy between the demonstrated and reproduced probabilistic trajectories. Following Ke et al. (2021) and Ghasemipour et al. (2020), we exploit tools from information theory and consider defining loss functions with f-divergences. Noticeably, a number of existing imitation learning algorithms, developed in the context of sequential decision-making or programming by demonstration, can be seen as the minimization of some f-divergence. For example, behavior cloning minimizes the Kullback-Leibler (KL) divergence (Pomerleau 1989), KMP minimizes the reverse KL divergence (Huang et al. 2019), and GAIL minimizes the Jensen-Shannon (JS) divergence (Ho and Ermon 2016). We adopt these ideas in the structured prediction framework and show that by using different divergences as loss functions, it is possible to obtain different imitation strategies, that we call *imitation modes*. Different imitation modes determine different coupling effects between the mean and the covariance of the obtained probabilistic trajectory policy.

A comparison between our approach and state-of-the-art algorithms is shown in Table 1. To summarize, our contribution is the development of a structured prediction framework for robot motion imitation that enables:

(i) Prediction of outputs of a variety of types, including Euclidean and manifold-structured trajectories;
(ii) Imitation of expert demonstrations with multiple imitation modes by means of different f-divergences;
(iii) Modulation of trajectories to adapt the learned motion skills to novel working settings.

The rest of the paper is organized as follows. In Section 2, we present the proposed algorithmic framework, casting imitation learning as a structured prediction problem and constructing loss functions based on f-divergences. Also, strategies for trajectory modulation are provided. Within the proposed framework, we introduce its practical implementation in Section 3, where both Euclidean and manifold-valued outputs are considered. The experimental results, including a comparison against previous algorithms and real-world tasks, are reported in Section 4 to show the effectiveness of the proposed approach. We review related work and discuss the limitations and future work in Section 5. Finally, we conclude the paper in Section 6.

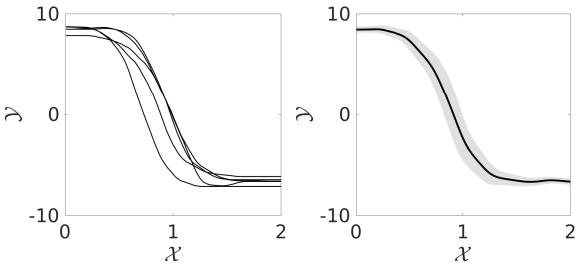


Figure 1. An illustrative example of data processing for probabilistic imitation learning. The *left* figure plots multiple demonstrations and the *right* figure plots the obtained probabilistic trajectory. The solid line represents the mean and the shallow area represents the covariance.

2 A Structured Prediction Approach to Probabilistic Imitation Learning

In this section, we first briefly review the background of probabilistic imitation learning and present our problem formulation (Section 2.1). We then present our algorithmic framework that reveals a structured approach to probabilistic imitation learning (Section 2.2), followed by trajectory modulation strategies (Section 2.3).

2.1 Background and Problem Setting

Probabilistic approaches are very popular in robot imitation learning (Billard et al. 2008). Compared to deterministic techniques, they can provide more information (such as variability and correlation) to a robot learner. To perform probabilistic imitation, a human teacher usually presents multiple demonstrations for a single task. Assume that the raw data collected from M demonstrations of fixed length N is formatted as $\{\{\mathbf{x}_n^m, \mathbf{y}_n^m\}_{n=1}^N\}_{m=1}^M$, where $\mathbf{x}_n^m \in \mathcal{X}$ is the input and $\mathbf{y}_n^m \in \mathcal{Y}$ denotes the output.

The data types of the input and the output spaces in robot imitation learning can vary in different application scenarios. For instance, in a common scenario where the robot needs to learn a time-indexed trajectory, we have $\mathcal{X} = \mathbb{R}$. In a more complex task such as human-robot collaboration, the robot could be required to react to the human's position. As a result, the input that the robot takes is a vector rather than a scalar, i.e., $\mathcal{X} = \mathbb{R}^{\mathcal{I}}$ with $\mathcal{I} > 1$ being the dimensionality. The output space $\mathcal{Y}$ is usually considered to have a Euclidean structure $\mathbb{R}^{\mathcal{O}}$ with $\mathcal{O}$ being the dimensionality. Additionally, often there are problems where manifold-type constraints are enforced on the output value, i.e., $\mathcal{Y} = \mathcal{M}$, where $\mathcal{M}$ denotes a manifold.

To exploit the probabilistic properties from raw demonstration datasets, suitable statistical learning tools such as mixture models (Billard et al. 2008) and their various extensions (Zeestraten et al. 2017b; Simo-Serra et al. 2017) can be employed. More precisely, we can have

$$\{\{\mathbf{x}_n^m, \mathbf{y}_n^m\}_{n=1}^N\}_{m=1}^M \xrightarrow[\text{processing}]{\text{data}} \mathbb{D} = \{\mathbf{x}_n, \tilde{\mathbf{y}}_n\}_{n=1}^N, \quad (1)$$

where the output $\tilde{\mathbf{y}}$ in the dataset $\mathbb{D}$ lies in a probability space as a result of multiple demonstrations and can be approximated by some Gaussian-like distribution $\mathcal{P}(\mathcal{Y})$. An illustrative example of data processing is shown in Figure 1.

In the context of robot movement imitation, a central topic is how to generate a trajectory so that a robot can mimic a demonstrator as closely as possible. To achieve this goal, the robot should behave as the demonstrator in response to a query input. It is then critical to find a suitable mapping between the input and output values based on the collected demonstrations (Zahra et al. 2022). This classical view on imitation learning is reminiscent of the objective of supervised learning: To find an input/output function given input/output pairs.

Focusing on probabilistic imitation learning, our goal is to address the motion imitation problem by learning the mapping rule $\mathbf{s} : \mathcal{X} \to \mathcal{P}(\mathcal{Y})$ given the dataset $\mathbb{D}$. Formally, the problem formulation can be described as

$$\text{find } \mathbf{s} : \mathcal{X} \to \mathcal{P}(\mathcal{Y}) \quad \text{given } \{\mathbf{x}_n, \tilde{\mathbf{y}}_n\}_{n=1}^N, \quad (2)$$

where we are looking for a mapping rule with outputs being probability distributions.

2.2 A Structured Prediction Perspective on Motion Imitation

Before addressing our concerned problem (2), we first recall the standard supervised learning setting

$$\text{find } \mathbf{s} : \mathcal{X} \to \mathcal{Y} \quad \text{given } \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N. \quad (3)$$

In particular, following (Ciliberto et al. 2016), we call (3) a structured prediction problem whenever the space $\mathcal{Y}$ does not have a linear structure, e.g., when $\mathcal{Y}$ is manifold.

2.2.1 Structured Prediction via Surrogate Approach

Due to the lack of linearity in the output space, structured prediction is a very challenging problem. Here, we resort to a surrogate solution that is common in classification (Mroueh et al. 2012) and apply it to the structured prediction problem as shown in (Ciliberto et al. 2016) by leveraging results for vector-valued kernel learning (Álvarez et al. 2012).

The key steps of the surrogate approach are sketched as follows:

1. *Encoding*. Design an encoding $\mathbf{c} : \mathcal{Y} \to \mathcal{H}$ to map the structured output space $\mathcal{Y}$ into a vector space $\mathcal{H}$.
2. *Surrogate learning*. Solve the learning problem in the surrogate space. This is achieved by first choosing a surrogate loss $\mathcal{L} : \mathcal{H} \times \mathcal{H} \to \mathbb{R}$ and then finding $\mathbf{g} : \mathcal{X} \to \mathcal{H}$ which minimizes the sum of errors $\mathcal{L}(\mathbf{c}(\mathbf{y}_n), \mathbf{g}(\mathbf{x}_n))$ given the surrogate dataset $\{\mathbf{x}_n, \mathbf{c}(\mathbf{y}_n)\}_{n=1}^N$.

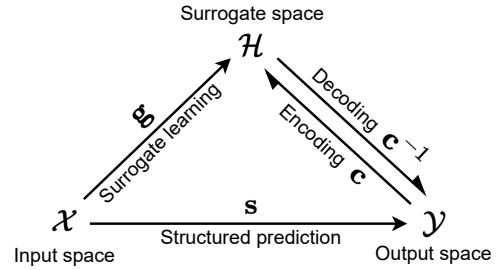


Figure 2. Schematic illustration of the surrogate approach to structured prediction.

3. *Decoding*. Recover $\mathbf{s}$ with a suitable decoding rule $\mathbf{c}^{-1} : \mathcal{H} \to \mathcal{Y}$, i.e., $\mathbf{s} = \mathbf{c}^{-1} \circ \mathbf{g} : \mathcal{X} \to \mathcal{Y}$.

A pictorial illustration of structured prediction by the surrogate framework is shown in Figure 2. Next, we discuss how the encoding steps can be performed *implicitly* for a wide spectrum of loss functions.

2.2.2 Implicit Encoding Framework As discussed in (Ciliberto et al. 2020), the embedding can be applied implicitly for the case of Structure Encoding Loss Functions (SELF) $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ for which there exists a separable Hilbert space $\mathcal{H}_\mathcal{Y}$ with inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}_\mathcal{Y}}$, a continuous feature map $\mathbf{c} : \mathcal{Y} \to \mathcal{H}_\mathcal{Y}$, and a continuous linear operator $V : \mathcal{H}_\mathcal{Y} \to \mathcal{H}_\mathcal{Y}$ such that for all $\mathbf{y}, \mathbf{y}' \in \mathcal{Y}$ we have

$$\Delta(\mathbf{y}, \mathbf{y}') = \langle \mathbf{c}(\mathbf{y}), V\mathbf{c}(\mathbf{y}') \rangle_{\mathcal{H}_\mathcal{Y}}. \tag{4}$$

To address the surrogate learning problem, we consider the following linearly parameterized model

$$\mathbf{g}(\mathbf{x}) = \mathbf{W}\boldsymbol{\varphi}(\mathbf{x}) \in \mathbb{R}^M, \tag{5}$$

where $\boldsymbol{\varphi} : \mathcal{X} \to \mathbb{R}^P$ denotes a feature map and $\mathbf{W} \in \mathbb{R}^{M \times P}$ denotes learnable parameters. The estimation of $\mathbf{W}$ can be determined by addressing the following multi-variate ridge regression problem:

$$\min_{\mathbf{W}} \frac{1}{N} \sum_{n=1}^{N} \|\mathbf{W}\boldsymbol{\varphi}(\mathbf{x}_n) - \mathbf{c}(\mathbf{y}_n)\|^2_{\mathcal{H}_\mathcal{Y}} + \lambda \|\mathbf{W}\|^2_F. \tag{6}$$

The solution to (6) can be shown to be

$$\widehat{\mathbf{W}} = \mathbf{C}(\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + N\lambda \mathbf{I}_N)^{-1} \boldsymbol{\Phi}^\top, \tag{7}$$

where we denote $\mathbf{C} = \big[\mathbf{c}(\mathbf{y}_1), \ldots, \mathbf{c}(\mathbf{y}_N)\big] \in \mathbb{R}^{M \times N}$ and $\boldsymbol{\Phi} = \big[\boldsymbol{\varphi}(\mathbf{x}_1), \ldots, \boldsymbol{\varphi}(\mathbf{x}_N)\big] \in \mathbb{R}^{P \times N}$. Also, $\lambda > 0$ is a regularization parameter and $\mathbf{I}_N \in \mathbb{R}^{N \times N}$ denotes the identity matrix of size N. Besides, $\|\cdot\|^2_F$ denotes the squared Frobenius norm of a matrix, i.e., the sum of all its squared elements.

By substituting (7) into (5), the solution to the surrogate learning problem is given by

$$\widehat{\mathbf{g}}(\mathbf{x}) = \widehat{\mathbf{W}}\boldsymbol{\varphi}(\mathbf{x}) = \sum_{n=1}^{N} \boldsymbol{\alpha}_n(\mathbf{x}) \mathbf{c}(\mathbf{y}_n), \tag{8}$$

where $\boldsymbol{\alpha}_n(\mathbf{x})$ is the n-th entry of $\boldsymbol{\alpha}(\mathbf{x}) \in \mathbb{R}^N$:

$$\boldsymbol{\alpha}(\mathbf{x}) = (\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + N\lambda \mathbf{I}_N)^{-1} \boldsymbol{\Phi}^\top \boldsymbol{\varphi}(\mathbf{x}) \tag{9}$$

$$= (\mathbf{K} + N\lambda \mathbf{I}_N)^{-1} \mathbf{k}_x, \tag{10}$$

where the kernel trick is invoked to obtain (10) from (9). More precisely, given a kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, we have $k(\mathbf{x}, \mathbf{x}') = \langle \boldsymbol{\varphi}(\mathbf{x}), \boldsymbol{\varphi}(\mathbf{x}') \rangle$. The empirical kernel matrix $\mathbf{K} \in \mathbb{R}^{N \times N}$ is constructed as $\mathbf{K}_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$ and $\mathbf{k}_x \in \mathbb{R}^N$ is the vector defined by $\mathbf{k}_x = \big[k(\mathbf{x}, \mathbf{x}_1), \ldots, k(\mathbf{x}, \mathbf{x}_N)\big]^\top$.

By exploiting the property of SELF, the decoder is designed such that the predictor has the form

$$\mathbf{s}(\mathbf{x}) = \operatorname*{argmin}_{\mathbf{y} \in \mathcal{Y}} \langle \mathbf{c}(\mathbf{y}), V\mathbf{g}(\mathbf{x}) \rangle_{\mathcal{H}_\mathcal{Y}}. \tag{11}$$

Finally, by plugging (8) into (11), we have

$$\widehat{\mathbf{s}}(\mathbf{x}) = \operatorname*{argmin}_{\mathbf{y} \in \mathcal{Y}} \left\langle \mathbf{c}(\mathbf{y}), V \left(\sum_{n=1}^{N} \boldsymbol{\alpha}_n(\mathbf{x}) \mathbf{c}(\mathbf{y}_n) \right) \right\rangle_{\mathcal{H}_\mathcal{Y}} \tag{12}$$

$$= \operatorname*{argmin}_{\mathbf{y} \in \mathcal{Y}} \sum_{n=1}^{N} \boldsymbol{\alpha}_n(\mathbf{x}) \Delta(\mathbf{y}, \mathbf{y}_n), \tag{13}$$

where we used the linearity property of the inner product and the definition of (4) to obtain (13) from (12).

In summary, when applying the implicit embedding framework to solve the structured prediction problem (3), the procedure consists of two steps:

1. *Surrogate learning*: Calculate the input-dependent weights $\boldsymbol{\alpha}$.
2. *Decoding*: Optimize the $\boldsymbol{\alpha}$-weighted linear combination of losses $\Delta(\mathbf{y}, \mathbf{y}_n)$.

The key insight is that the encoding rule $\mathbf{c}$ and the surrogate space $\mathcal{H}_\mathcal{Y}$ are no longer explicitly needed and in this sense the encoding is implicit.

On the basis of the implicit encoding framework (Ciliberto et al. 2020), in the following we present the development of the proposed imitation learning algorithm. We first outline the main idea of performing probabilistic trajectory imitation via structured prediction (Section 2.2.3). Afterwards, we show the strategies for trajectory modulation, which is essential for robots to reproduce motion skills in environments different from the one experienced during the demonstrations (Section 2.3). Then, we illustrate motion imitation in the case of Euclidean and Riemannian probabilistic trajectories (Section 3).

2.2.3 Loss Function Design by f-Divergence We now consider addressing problem (2) where the outputs are trajectory distributions in a probability space $\mathcal{P}$. Given an input $\mathbf{x}$, an immediate application of the solution to structured prediction (13) becomes

$$\widehat{\mathbf{s}}(\mathbf{x}) = \operatorname*{argmin}_{\tilde{\mathbf{y}} \in \mathcal{P}} \sum_{n=1}^{N} \boldsymbol{\alpha}_n(\mathbf{x}) \Delta(\tilde{\mathbf{y}}, \tilde{\mathbf{y}}_n), \tag{14}$$

where a suitable loss function needs to be designed to quantify the discrepancy between two probability distributions $\tilde{\mathbf{y}}$ and $\tilde{\mathbf{y}}_n$ in $\mathcal{P}$. We propose to leverage tools from an information-theoretic perspective and choose the family of f-divergences provided the f-divergences generalize similarity measures between probability distributions. Such a choice is also in alignment with the findings from sequential decision-making that imitation learning can be treated as

the divergence minimization between expert and learner trajectory distributions (Ke et al. 2021).

The expression for the f-divergence is given as*

$$D_f\left(\tilde{\mathbf{y}}_n(\mathbf{x}), \tilde{\mathbf{y}}(\mathbf{x})\right) \triangleq \mathbb{E}_{\tilde{\mathbf{y}}(\mathbf{x})}\left[f\left(\frac{d\tilde{\mathbf{y}}_n(\mathbf{x})}{d\tilde{\mathbf{y}}(\mathbf{x})}\right)\right], \tag{15}$$

where $f : R^+ \to R$ represents a convex function with $f(1) = 0$. By choosing different functions f, a broad class of divergences can be defined. Common examples include the KL divergence, the reverse KL divergence, and the Jensen-Shannon divergence (see Pardo (2018) for a full list). Given that different types of f-divergence will result in different imitation policies, we then refer to these different imitation policies as *imitation modes*.

Finally, by substituting (15) into (14), the estimator for probabilistic trajectory prediction is obtained as

$$\widehat{\mathbf{s}}(\mathbf{x}) = \underset{\tilde{\mathbf{y}}\in\mathcal{P}}{\texttt{argmin}} \sum_{n=1}^{N} \boldsymbol{\alpha}_n(\mathbf{x}) \mathbb{E}_{\tilde{\mathbf{y}}(\mathbf{x})}\left[f\left(\frac{d\tilde{\mathbf{y}}_n(\mathbf{x})}{d\tilde{\mathbf{y}}(\mathbf{x})}\right)\right]. \tag{16}$$

2.3 Trajectory Modulation

As the demonstration and reproduction environments can differ, robots should be able to adapt the learned motor skills during reproduction. To this aim, it is essential to endow robots with adaptability to the arising requirements through spatial or temporal trajectory modulation.

For example, trajectory modulation for desired *via-points* can be used for reaching a region of interest or avoiding collisions. Also, the capability of setting off from a new *start point* or converging to a different *end point* can make robots more flexible in tasks like pushing or pick-and-place. Our approach guarantees that movements can be adapted on the fly during the execution of Euclidean trajectories. Moreover, to generate more complex behaviors, multiple movement trajectories can be co-activated simultaneously (Duan et al. 2019). Such concurrent co-activation of different trajectories is also known as trajectory *superposition*, which can significantly improve motion expressiveness.

Besides spatial modulation, temporal modulation is also a necessary capability for tasks that are sensitive to correct timing. By speeding up or slowing down the robot's movement, trajectories can then be temporally adapted for striking-based manipulation or walking speed adjustment in locomotion. Temporal modulation can also be used to avoid time-dependent collisions.

2.3.1 Spatial Modulation

We consider the issue of passing through additional desired *via-points*. Assume that there are J new desired points stored in the dataset $\mathbb{D}_v = \{\mathbf{x}_j, \tilde{\mathbf{y}}_j\}_{j=1}^{J}$ with each one repeating $w_j > 1$ times. To take these new requirements into account, we concatenate the new desired dataset to the original demonstrated one. Consequently, the updated dataset to train our estimator now becomes $\mathbb{D} \cup \mathbb{D}_v$ with a total number of points $N' = N + \sum_j^J w_j$. Afterwards, (14) should be applied to the new dataset.

It is noteworthy that the size of the kernel matrix now increases to $\mathbb{R}^{N'\times N'}$. Owing to the computational burden incurred by matrix inversion, it will be favorable to reduce the size of the kernel matrix. To this end, we examine surrogate learning in the deterministic setting. The optimization problem (6) incorporating weighted terms is now formulated as

$$\min_{\mathbf{W}} \frac{1}{N'} \sum_{n=1}^{N+J} w_n \|\mathbf{W}\boldsymbol{\varphi}(\mathbf{x}_n) - \mathbf{c}(\mathbf{y}_n)\|^2_{\mathcal{H}_\mathcal{Y}} + \lambda\|\mathbf{W}\|^2_F \tag{17}$$

where each weight w_n is defined as

$$w_n = \begin{cases} w_j & \mathbf{x}_n \in \mathbb{D}_v, \\ 1 & \text{otherwise.} \end{cases} \tag{18}$$

The estimator for trajectory adaptation can still be expressed similarly to (14), except that the size of the kernel matrix is reduced to $\mathbb{R}^{(N+J)\times(N+J)}$, which will yield faster computation speed when carrying out matrix inversion. Furthermore, the coefficients are written as

$$\boldsymbol{\alpha}'(\mathbf{x}) = (\mathbf{K}' + N'\lambda\mathbf{I}_{N+J})^{-1}\mathbf{k}'_x, \tag{19}$$

where $\mathbf{K}'$ and $\mathbf{k}'_x$ are obtained by weighing the rows of $\mathbf{K}$ and $\mathbf{k}_x$ that involve $\mathbf{x}_j$ by w_j.

As for trajectory *superposition*, the robot is expected to follow H prioritized trajectories. We denote the related dataset by $\mathbb{D}_s = \{w_h, \{\mathbf{x}_n^h, \tilde{\mathbf{y}}_n^h\}_{n=1}^{N}\}_{h=1}^{H}$ with priorities normalized, i.e., $\sum_{h=1}^{H} w_h = 1$. Given the assigned priorities, we construct the loss function by weighing the individual loss evaluations as:

$$\Delta(\tilde{\mathbf{y}}, \tilde{\mathbf{y}}_n) = \sum_{h=1}^{H} w_h D_f(\tilde{\mathbf{y}}_n^h, \tilde{\mathbf{y}}), \tag{20}$$

which results in the estimator as

$$\widehat{\mathbf{s}}(\mathbf{x}) = \underset{\tilde{\mathbf{y}}\in\mathcal{P}}{\texttt{argmin}} \sum_{n=1}^{N} \boldsymbol{\alpha}_n(\mathbf{x}) \sum_{h=1}^{H} w_h D_f(\tilde{\mathbf{y}}_n^h, \tilde{\mathbf{y}}). \tag{21}$$

The algorithmic framework of the proposed imitation learning approach incorporating spatial trajectory modulation is summarized in Algorithm 1.

Algorithm 1: Imitation Learning by Structured Prediction

Initialization:
Retrieve dataset $\mathbb{D}$ from demonstrations;
Define kernel k and hyperparameter λ;
Choose imitation mode f;
Trajectory modulation:
Specify desired points in $\mathbb{D}_v$;
Prioritize trajectories in $\mathbb{D}_s$;
Aggregate dataset as $\mathbb{D} \cup \mathbb{D}_v \cup \mathbb{D}_s$;
Motion generation:
Input: query point $\mathbf{x}$;
Calculate weights $\boldsymbol{\alpha}'(\mathbf{x})$;
Output: estimated value $\widehat{\mathbf{s}}(\mathbf{x})$;

*We put "true" distribution first to comply with forward KL divergence.

2.3.2 Temporal Modulation When dealing with temporal modulation, a phase variable z can be introduced to decouple the dependence from time (Ijspeert et al. 2013). The choice of phase $z(t)$ can be any monotonic increasing function with respect to the time stamp t. By making the movement depend on the phase rather than time, a faster or slower execution of the movement is then permitted. Therefore, the desired temporal evolution of the movement can be achieved by tuning the rate of the phase variable. A common choice of the monotonic function is first-order linear dynamics (Ijspeert et al. 2013).

3 Practical Implementation of the Algorithm

We begin by noting that Algorithm 1 is a meta-algorithm since it requires solving an optimization problem over the probability space. In this section, we will illustrate how to practically deploy the algorithm. Towards this end, we make a specific choice that each output distribution satisfies a Gaussian $\tilde{\mathbf{y}}_n \sim \mathcal{N}_\alpha(\boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n)$ (on a Euclidean $\mathcal{N}$ or a manifold $\mathcal{N}_\mathcal{M}$ (Zeestraten et al. 2017b)) with mean $\boldsymbol{\mu}_n$ and covariance $\boldsymbol{\Sigma}_n$. As a result, the dataset has a format of $\mathbb{D}_\mathcal{N} = \{\mathbf{x}_n, (\boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n)\}_{n=1}^N$.

Our intention here is to find separate estimators $\mathbf{s}_m$ and $\mathbf{s}_c$ to predict mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$, respectively. Particularly, given a query point $\mathbf{x}$, the corresponding output becomes $\tilde{\mathbf{y}}(\mathbf{x}) \sim \mathcal{N}_\alpha(\mathbf{s}_m(\mathbf{x}), \mathbf{s}_c(\mathbf{x}))$. To find these estimators, we instantiate the template of problem (16) by choosing a specific f-divergence function and restricting the type of probabilistic trajectories to Gaussian distributions. Consequently, problem (16) becomes

$$\widehat{\mathbf{s}}(\mathbf{x}) = \underset{(\boldsymbol{\mu},\boldsymbol{\Sigma})\in\mathcal{Y}\times\mathcal{Y}^2}{\texttt{argmin}} \sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x}) D_f(\mathcal{N}_\alpha(\boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n), \mathcal{N}_\alpha(\boldsymbol{\mu}, \boldsymbol{\Sigma})). \tag{22}$$

The optimization problem (22) can be solved by taking the derivatives with respect to the design variables $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, respectively, and then setting the obtained derivatives to zero. Therefore, we need to calculate

$$\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x}) \frac{\partial D_f(\boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{\partial \boldsymbol{\mu}} = \mathbf{0}, \tag{23}$$

$$\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x}) \frac{\partial D_f(\boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{\partial \boldsymbol{\Sigma}} = \mathbf{0}. \tag{24}$$

When calculating the partial derivatives as required by (23) and (24), the cost terms containing $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ shall be singled out, separately. The estimators of mean $\widehat{\mathbf{s}}_m$ and covariance $\widehat{\mathbf{s}}_c$ will then appear as

$$\widehat{\mathbf{s}}_m(\mathbf{x}) = \underset{\boldsymbol{\mu}\in\mathcal{Y}}{\texttt{argmin}} \sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x}) \Delta_m(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \tag{25}$$

$$\widehat{\mathbf{s}}_c(\mathbf{x}) = \underset{\boldsymbol{\Sigma}\in\mathcal{Y}^2}{\texttt{argmin}} \sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x}) \Delta_c(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \tag{26}$$

where Δ_m groups all the terms containing $\boldsymbol{\mu}$ for mean prediction and Δ_c groups all the terms containing $\boldsymbol{\Sigma}$ for covariance prediction. We drop the dependence of the cost terms on n without ambiguity. In addition, when predicting the covariance matrix, the result needs to be restricted to the cone of symmetric positive semi-definite matrices.

Next, we discuss the realization of different imitation modes by choosing different f-divergences in Section 3.1. We show in Section 3.2 how to deal with manifold-valued probabilistic trajectories.

3.1 Imitation with Euclidean-Valued Output

As discussed in Section 2.2.3, different choices of f can result in different imitation strategies $\tilde{\mathbf{y}} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ to imitate the expert policy $\tilde{\mathbf{y}}_n \sim \mathcal{N}(\boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n)$ in response to a query input $\mathbf{x}$. We called these strategies imitation modes. Here we consider the case where the trajectory mean lies in a Euclidean space. Specifically, we show how different imitation learning algorithms can be obtained by exploiting two common f-divergences, namely the KL divergence and the reverse KL divergence.

3.1.1 KL divergence We start with the well-known KL-divergence to illustrate the motivation for the cost functions design. The KL divergence is obtained by taking $f(u) = u\log(u)$. Using the properties of the KL divergence between two multivariate Gaussian distributions, the corresponding loss function is given by

$$\begin{aligned} D_{\texttt{KL}}(\tilde{\mathbf{y}}_n, \tilde{\mathbf{y}}) &= \mathbb{E}_{\tilde{\mathbf{y}}}\left[\frac{d\tilde{\mathbf{y}}_n}{d\tilde{\mathbf{y}}} \log\left(\frac{d\tilde{\mathbf{y}}_n}{d\tilde{\mathbf{y}}}\right)\right] \\ =\frac{1}{2}\Big(&\underbrace{\underbrace{(\boldsymbol{\mu}-\boldsymbol{\mu}_n)^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}-\boldsymbol{\mu}_n)}_{\Delta_m} + \log|\boldsymbol{\Sigma}| + \texttt{Tr}(\boldsymbol{\Sigma}^{-1}\boldsymbol{\Sigma}_n)}_{\Delta_c} \\ &- \log|\boldsymbol{\Sigma}_n| - \texttt{dim}(\mathcal{Y})\Big), \end{aligned} \tag{27}$$

where $|\cdot|$ denotes the determinant of a matrix, $\texttt{Tr}(\cdot)$ denotes the trace of a matrix, and $\texttt{dim}(\mathcal{Y})$ indicates the dimensionality of the output space.

Grouping the terms that include $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, respectively, we can acquire the expressions for the optimization cost functions Δ_m and Δ_c from (27). Then we can plug these expressions in (25) and (26) so that the estimators with the KL-divergence imitation mode are given by

$$\widehat{\mathbf{s}}_m(\mathbf{x}) = \underset{\boldsymbol{\mu}\in\mathcal{Y}}{\texttt{argmin}} \sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})\big((\boldsymbol{\mu}-\boldsymbol{\mu}_n)^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}-\boldsymbol{\mu}_n)\big), \tag{28}$$

$$\begin{aligned} \widehat{\mathbf{s}}_c(\mathbf{x}) = \underset{\boldsymbol{\Sigma}\in\mathcal{Y}^2}{\texttt{argmin}} \sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})\big(&(\boldsymbol{\mu}-\boldsymbol{\mu}_n)^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}-\boldsymbol{\mu}_n) \\ &+ \log|\boldsymbol{\Sigma}| + \texttt{Tr}(\boldsymbol{\Sigma}^{-1}\boldsymbol{\Sigma}_n)\big). \end{aligned} \tag{29}$$

To compute the optimal mean and covariance predictions, we set to zero the derivatives of (28) and (29) with respect to $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, which yields

$$\boldsymbol{\mu} = \frac{\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})\boldsymbol{\mu}_n}{\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})}, \tag{30}$$

Table 2. List of imitation modes based on different divergences

	Kullback-Leibler divergence	Reverse Kullback-Leibler divergence
$f(u)$	$u\log(u)$	$-\log(u)$
$\Delta_m(\boldsymbol{\mu},\boldsymbol{\Sigma})$	$(\boldsymbol{\mu}-\boldsymbol{\mu}_n)^\top\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}-\boldsymbol{\mu}_n)$	$(\boldsymbol{\mu}-\boldsymbol{\mu}_n)^\top\boldsymbol{\Sigma}_n^{-1}(\boldsymbol{\mu}-\boldsymbol{\mu}_n)$
$\widehat{\mathbf{s}}_m(\mathbf{x})$	$\frac{\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})\boldsymbol{\mu}_n}{\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})}$	$\big(\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})\boldsymbol{\Sigma}_n^{-1}\big)^{-1}\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})\boldsymbol{\Sigma}_n^{-1}\boldsymbol{\mu}_n$
$\Delta_c(\boldsymbol{\mu},\boldsymbol{\Sigma})$	$(\boldsymbol{\mu}-\boldsymbol{\mu}_n)^\top\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}-\boldsymbol{\mu}_n)+\log\lvert\boldsymbol{\Sigma}\rvert+\mathtt{Tr}(\boldsymbol{\Sigma}^{-1}\boldsymbol{\Sigma}_n)$	$-\log\lvert\boldsymbol{\Sigma}\rvert+\mathtt{Tr}(\boldsymbol{\Sigma}_n^{-1}\boldsymbol{\Sigma})$
$\widehat{\mathbf{s}}_c(\mathbf{x})$	$\frac{\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})\big((\boldsymbol{\mu}-\boldsymbol{\mu}_n)(\boldsymbol{\mu}-\boldsymbol{\mu}_n)^\top+\boldsymbol{\Sigma}_n\big)}{\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})}$	$\left(\frac{\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})\boldsymbol{\Sigma}_n^{-1}}{\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})}\right)^{-1}$

$$\boldsymbol{\Sigma}=\frac{\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})\big((\boldsymbol{\mu}-\boldsymbol{\mu}_n)(\boldsymbol{\mu}-\boldsymbol{\mu}_n)^\top+\boldsymbol{\Sigma}_n\big)}{\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})} \tag{31}$$

$$\approx\frac{\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})\boldsymbol{\Sigma}_n}{\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})}. \tag{32}$$

We note that the prediction of the covariance matrix according to (31) is dependent on the predicted mean value given by (30). By contrast, the prediction of the mean value is only dependent on the training output mean $\boldsymbol{\mu}_n$. Therefore, we should predict the mean first before computing the covariance prediction. In addition, if we would like to alleviate the interference from the mean value on the covariance prediction, it can be considered to approximate (31) by omitting the term $(\boldsymbol{\mu}-\boldsymbol{\mu}_n)(\boldsymbol{\mu}-\boldsymbol{\mu}_n)^\top$ as in (32).

3.1.2 Reverse KL divergence Besides the KL divergence, there are plenty of other divergences that can be used to construct loss functions. For example, we here consider the reverse KL divergence, which reflects the asymmetry of the KL divergence. The reverse KL divergence is defined by $f(u)=-\log(u)$, which gives rise to the associated loss function as

$$\begin{aligned}D_{\mathtt{RKL}}(\tilde{\mathbf{y}}_n,\tilde{\mathbf{y}})&=\mathbb{E}_{\tilde{\mathbf{y}}}\left[\log\left(\frac{d\tilde{\mathbf{y}}}{d\tilde{\mathbf{y}}_n}\right)\right]\\&=\frac{1}{2}\Big(\underbrace{(\boldsymbol{\mu}-\boldsymbol{\mu}_n)^\top\boldsymbol{\Sigma}_n^{-1}(\boldsymbol{\mu}-\boldsymbol{\mu}_n)}_{\Delta_m}\underbrace{-\log\lvert\boldsymbol{\Sigma}\rvert+\mathtt{Tr}(\boldsymbol{\Sigma}_n^{-1}\boldsymbol{\Sigma})}_{\Delta_c}\\&\quad+\log\lvert\boldsymbol{\Sigma}_n\rvert-\mathtt{dim}(\mathcal{Y})\Big).\end{aligned} \tag{33}$$

Similarly to Section 3.1.1, we collect all the terms involving $\boldsymbol{\mu}$ to specify the cost function Δ_m for mean prediction and collect all the terms involving $\boldsymbol{\Sigma}$ to specify the cost function Δ_c for covariance prediction. By plugging the cost functions given by (33) into (25) and (26), the reverse-KL imitation mode estimators can be expressed as

$$\widehat{\mathbf{s}}_m(\mathbf{x})=\underset{\boldsymbol{\mu}\in\mathcal{Y}}{\mathtt{argmin}}\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})\big((\boldsymbol{\mu}-\boldsymbol{\mu}_n)^\top\boldsymbol{\Sigma}_n^{-1}(\boldsymbol{\mu}-\boldsymbol{\mu}_n)\big), \tag{34}$$

$$\widehat{\mathbf{s}}_c(\mathbf{x})=\underset{\boldsymbol{\Sigma}\in\mathcal{Y}^2}{\mathtt{argmin}}\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})\big(-\log\lvert\boldsymbol{\Sigma}\rvert+\mathtt{Tr}(\boldsymbol{\Sigma}_n^{-1}\boldsymbol{\Sigma})\big). \tag{35}$$

The solutions are also calculated by setting the derivatives of (34) and (35) with respect to the design variables $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ equal to zero, respectively. Therefore, the optimal predictions for mean and covariance are given by

$$\boldsymbol{\mu}=\left(\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})\boldsymbol{\Sigma}_n^{-1}\right)^{-1}\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})\boldsymbol{\Sigma}_n^{-1}\boldsymbol{\mu}_n, \tag{36}$$

$$\boldsymbol{\Sigma}=\left(\frac{\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})\boldsymbol{\Sigma}_n^{-1}}{\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})}\right)^{-1}. \tag{37}$$

It can be seen that the prediction of the mean value by (36) relies on both the training output covariance $\boldsymbol{\Sigma}_n$ and mean $\boldsymbol{\mu}_n$. By contrast, the prediction of the covariance matrix in (37) only depends on the training output covariance. As a side note, the predicted covariance matrix given by (37) can be utilized for mean prediction to avoid repetitive computation of the term $(\sum_{n=1}^N \boldsymbol{\alpha}_n(\mathbf{x})\boldsymbol{\Sigma}_n^{-1})^{-1}$.

As a sanity check, it can be seen that the predicted covariance matrix in (31) and (37) is indeed symmetric positive semi-definite if the training covariance matrices are symmetric positive definite since (i) the outer product of a vector, (ii) the inverse of a symmetric positive definite matrix, and (iii) the sum of the symmetric positive semi-definite matrices are all symmetric positive semi-definite matrices. While the weight $\boldsymbol{\alpha}_n(\mathbf{x})$ can be non-positive for some terms, in practice, its score at the proximity to the input query point is positive and will usually dominate other terms. For other possible choices of f, the constraint for $\boldsymbol{\Sigma}$ being a symmetric positive semi-definite matrix should be taken into account explicitly.

It should be noted that the optimization problem (22) is derived on the basis of the implicit embedding framework. Therefore, to make the formulation valid, it is important for the divergence-inspired loss functions $D_{\mathtt{KL}}$ (27) and $D_{\mathtt{RKL}}$ (33) to be SELF. In fact, it is possible to show these are the cases. We provide a proof sketch in Appendix A.

Our main results are summarized in Table 2 and the algorithm for imitation of Euclidean-valued probabilistic trajectories is written in Algorithm 2.

3.1.3 Learning the Velocity Profile of Temporal Trajectories In tasks such as throwing and catching, robots need to exhibit dynamic behavior, which requires learning motor skills involving not only position but also velocity or even higher-order derivatives. However, a direct application of the structured prediction method developed so far could be infeasible, since both mean prediction strategies given by (30) and (36) are not aware of the constraint on the

Algorithm 2: Imitation with Euclidean-Valued Output

Collect trajectories from multiple demonstrations;
`/* Assumption of Gaussian output */`
Process raw data for $\mathbb{D}_{\mathcal{N}} = \{\mathbf{x}_n, (\boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n)\}_{n=1}^{N}$;
Define the kernel k and parameter λ;
Choose imitation mode f;
switch *imitation mode* f **do**
  for $\mathbf{x} = \mathbf{x}_{\text{Start}}, \ldots, \mathbf{x}_{\text{End}}$ **do**
    Input: a query point $\mathbf{x}$;
    Calculate the weights $\boldsymbol{\alpha}(\mathbf{x})$;
    case $f(u) = u\log(u)$ **do**
      `/* Predict with the KL divergence mode */`
      Output: mean $\boldsymbol{\mu}$ as per (30);
      Output: covariance $\boldsymbol{\Sigma}$ as per (31);
    case $f(u) = -\log(u)$ **do**
      `/* Predict with the reverse KL divergence mode */`
      Output: mean $\boldsymbol{\mu}$ as per (36);
      Output: covariance $\boldsymbol{\Sigma}$ as per (37);
    otherwise do
      Formulate Δ_m and Δ_c motivated by (22);
      Output: mean $\boldsymbol{\mu}$ by optimizing (25);
      Output: covariance $\boldsymbol{\Sigma}$ by optimizing (26);
      `// s.t.` $\boldsymbol{\Sigma} = \boldsymbol{\Sigma}^{\top}$ and $\boldsymbol{\Sigma} \succeq 0$

derivative relationship between position and velocity outputs. Therefore, we need to explicitly cope with the issue of learning temporal trajectories. We denote the output value $\mathbf{y}$ composed of both position $\boldsymbol{\mu}$ and velocity $\dot{\boldsymbol{\mu}}$ as

$$\mathbf{y}(t) = \begin{bmatrix} \boldsymbol{\mu}(t) \\ \dot{\boldsymbol{\mu}}(t) \end{bmatrix}. \tag{38}$$

Recall that we used the least-squares estimator (6) to solve the surrogate problem. As this treatment can not take the underlying temporal constraint of output values into account, a derivative constraint-aware estimator $\mathbf{g}$ is needed such that it can capture the temporal relationship.

We motivate our design for the estimation value $\widehat{\mathbf{g}}$ by referring to an analytic insight of the feature map. Specifically, to complement (5) for velocity learning, we employ the following parametric form

$$\mathbf{g}(t) = \begin{bmatrix} \mathbf{W}^{\top}\boldsymbol{\varphi}(t) \\ \mathbf{W}^{\top}\dot{\boldsymbol{\varphi}}(t) \end{bmatrix} \tag{39}$$

where the same weight matrix $\mathbf{W}$ is used for both velocity and position feature maps so that the time derivative constraint can be respected. The feature maps for velocity learning are expressed by

$$\dot{\boldsymbol{\varphi}}(t) = \lim_{\delta \to 0} \frac{\boldsymbol{\varphi}(t+\delta) - \boldsymbol{\varphi}(t-\delta)}{2\delta}. \tag{40}$$

The estimated value for $\mathbf{W}$ can be obtained by solving the following optimization problem

$$\min_{\mathbf{W}} \frac{1}{N}\sum_{n=1}^{N} \left\| \begin{bmatrix} \boldsymbol{\varphi}_n^{\top} \\ \dot{\boldsymbol{\varphi}}_n^{\top} \end{bmatrix} \mathbf{W} - \begin{bmatrix} \boldsymbol{\mu}_n^{\top} \\ \dot{\boldsymbol{\mu}}_n^{\top} \end{bmatrix} \right\|_F^2 + \lambda \|\mathbf{W}\|_F^2. \tag{41}$$

By setting the derivative of (41) with respect to $\mathbf{W}$ to zero, we then obtain

$$\widehat{\mathbf{W}} = \boldsymbol{\Phi}(\boldsymbol{\Phi}^{\top}\boldsymbol{\Phi} + N\lambda\mathbf{I})^{-1}\mathbf{Y}, \tag{42}$$

where we represent $\boldsymbol{\Phi} = \left[\boldsymbol{\varphi}_1, \dot{\boldsymbol{\varphi}}_1, \ldots, \boldsymbol{\varphi}_N, \dot{\boldsymbol{\varphi}}_N\right]$ and $\mathbf{Y} = \left[\boldsymbol{\mu}_1, \dot{\boldsymbol{\mu}}_1, \ldots, \boldsymbol{\mu}_N, \dot{\boldsymbol{\mu}}_N\right]^{\top}$. Consequently, by substituting (42) into (39), we have the predicted values at time step t as

$$\widehat{\mathbf{g}}(t) = \begin{bmatrix} \mathbf{Y}^{\top}(\mathbf{K} + N\lambda\mathbf{I})^{-1}\mathbf{k}^p \\ \mathbf{Y}^{\top}(\mathbf{K} + N\lambda\mathbf{I})^{-1}\mathbf{k}^v \end{bmatrix} \tag{43}$$

where the vectors $\mathbf{k}^p$ and $\mathbf{k}^v$ are given by

$$\mathbf{k}_i^p = \begin{cases} \boldsymbol{\varphi}_i^{\top}\boldsymbol{\varphi}(t) \\ \dot{\boldsymbol{\varphi}}_i^{\top}\boldsymbol{\varphi}(t) \end{cases} \quad \mathbf{k}_i^v = \begin{cases} \boldsymbol{\varphi}_i^{\top}\dot{\boldsymbol{\varphi}}(t) & i = 2n-1, \\ \dot{\boldsymbol{\varphi}}_i^{\top}\dot{\boldsymbol{\varphi}}(t) & i = 2n. \end{cases} \tag{44}$$

The kernel matrix $\mathbf{K}$ is constructed as

$$\mathbf{K}_{i,j} = \begin{cases} \boldsymbol{\varphi}_i^{\top}\boldsymbol{\varphi}_j & i = 2n-1, \quad j = 2n'-1, \\ \boldsymbol{\varphi}_i^{\top}\dot{\boldsymbol{\varphi}}_j & i = 2n-1, \quad j = 2n', \\ \dot{\boldsymbol{\varphi}}_i^{\top}\boldsymbol{\varphi}_j & i = 2n, \quad j = 2n'-1, \\ \dot{\boldsymbol{\varphi}}_i^{\top}\dot{\boldsymbol{\varphi}}_j & i = 2n, \quad j = 2n', \end{cases} \tag{45}$$

where n and $n' = 1, 2, 3, \ldots, N$. After substituting $\dot{\boldsymbol{\varphi}}$ with its definition (40) into (45), we apply the kernel trick for the matrix as also shown by (Huang et al. 2019). In summary, the definition for the kernel matrix is given in Table 3, where $t^{\pm} \triangleq t \pm \delta$. Also, the kernelized version for (44) can be derived similarly.

Table 3. Definition for kernel matrix $\mathbf{K}$.

$\mathbf{K}_{i,j}$	$i = 2n-1$	$i = 2n$
$j = 2n'-1$	$k(t_i, t_j)$	$(k(t_i^+, t_j) - k(t_i^-, t_j))/2\delta$
$j = 2n'$	$(k(t_i, t_j^+) - k(t_i, t_j^-))/2\delta$	$(k(t_i^+, t_j^+) - k(t_i^+, t_j^-) - k(t_i^-, t_j^+) + k(t_i^-, t_j^-))/4\delta^2$

It should be noted that despite the presence of the time derivative constraint in (38), it still admits the Euclidean metric for such output values. Hence, there is no need for an encoding rule, as evident by (41) where the temporal trajectory outputs are directly used as the learning objectives instead of their embedding values. With the collapse of the surrogate space, the structured prediction estimator $\widehat{\mathbf{s}}(t)$ thus coincides with the mapping of surrogate learning (43).

Furthermore, to align with the formation of (30), we rewrite the solution to our temporal estimator $\widehat{\mathbf{s}}(t)$ towards the formalism of a weighted sum of output values:

$$\widehat{\mathbf{s}}(t) = \sum_{n=1}^{N} \boldsymbol{\alpha}_n \mathbf{y}_n. \tag{46}$$

The weight matrix $\boldsymbol{\alpha}_n$ is defined as

$$\boldsymbol{\alpha}_n = \begin{bmatrix} \boldsymbol{\alpha}_{2n-1}^p \mathbf{I} & \boldsymbol{\alpha}_{2n}^p \mathbf{I} \\ \boldsymbol{\alpha}_{2n-1}^v \mathbf{I} & \boldsymbol{\alpha}_{2n}^v \mathbf{I} \end{bmatrix} \tag{47}$$

where we have

$$\boldsymbol{\alpha}^p = (\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{k}^p \text{ and } \boldsymbol{\alpha}^v = (\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{k}^v. \tag{48}$$

To modulate the temporal trajectory such that it can pass through some desired via-point and/or via-velocity, $\mathbf{K}$, $\mathbf{k}^p$, and $\mathbf{k}^v$ need to be modified similarly to (19), i.e., the rows that involve desired adaptive behavior need to be weighed accordingly.

3.2 Imitation with Manifold-Valued Output

In this section, we present our algorithmic framework in the context of manifold structured prediction, i.e., when the output space $\mathcal{Y}$ is a manifold $\mathcal{M}$. Differently from Section 3.1, the learner's and the expert's probabilistic trajectories are now constrained to lie on a Riemannian manifold[†]. In this case, we consider the Riemannian Gaussian $\mathcal{N}_{\mathcal{M}}$ to represent the policy, which is usually approximated as (see e.g., Simo-Serra et al. (2017); Zeestraten et al. (2017b)):

$$\mathcal{N}_{\mathcal{M}}(\mathbf{y};\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}\mathtt{Log}_{\boldsymbol{\mu}}(\mathbf{y})^\top \boldsymbol{\Sigma}^{-1}\mathtt{Log}_{\boldsymbol{\mu}}(\mathbf{y})} \quad (49)$$

where $\boldsymbol{\mu}\in\mathcal{M}$ is the Riemannian mean and $\boldsymbol{\Sigma}$ is the covariance matrix defined in the tangent space $\mathcal{T}_{\boldsymbol{\mu}}\mathcal{M}$.

To construct the corresponding loss function, we also propose to compute the f-divergence between the learner policy $\tilde{\mathbf{y}}\sim\mathcal{N}_{\mathcal{M}}(\boldsymbol{\mu},\boldsymbol{\Sigma})$ and the expert policy $\tilde{\mathbf{y}}_n\sim\mathcal{N}_{\mathcal{M}}(\boldsymbol{\mu}_n,\boldsymbol{\Sigma}_n)$. Here, we take the KL divergence as an example, so that the loss function is

$$D_{\mathrm{KL}}(\tilde{\mathbf{y}}_n,\tilde{\mathbf{y}}) = \frac{1}{2}\int\Bigg(\log\frac{|\boldsymbol{\Sigma}|}{|\boldsymbol{\Sigma}_n|} - \mathtt{Log}_{\boldsymbol{\mu}_n}(\mathbf{y})\boldsymbol{\Sigma}_n^{-1}\mathtt{Log}_{\boldsymbol{\mu}_n}(\mathbf{y}) + \mathtt{Log}_{\boldsymbol{\mu}}(\mathbf{y})\boldsymbol{\Sigma}^{-1}\mathtt{Log}_{\boldsymbol{\mu}}(\mathbf{y})\Bigg)\tilde{\mathbf{y}}_n\,d\mathbf{y}. \quad (50)$$

To ease the computation of (50), we make use of the following approximation

$$\mathtt{Log}_{\boldsymbol{\mu}}(\mathbf{y}) \approx \mathtt{Log}_{\boldsymbol{\mu}}(\boldsymbol{\mu}_n) + \mathtt{Log}_{\boldsymbol{\mu}_n}(\mathbf{y}). \quad (51)$$

As a result, (50) can be approximated as

$$\frac{1}{2}\Big(\underbrace{\mathtt{Log}_{\boldsymbol{\mu}}(\boldsymbol{\mu}_n)^\top\boldsymbol{\Sigma}^{-1}\mathtt{Log}_{\boldsymbol{\mu}}(\boldsymbol{\mu}_n)}_{\Delta_m} + \underbrace{\log|\boldsymbol{\Sigma}| + \mathtt{Tr}(\boldsymbol{\Sigma}^{-1}\boldsymbol{\Sigma}_n)}_{\Delta_c} - \log|\boldsymbol{\Sigma}_n| - \mathtt{dim}(\mathcal{Y})\Big). \quad (52)$$

In view of the complex form of Δ_m, we propose to ease the computations by omitting the weight $\boldsymbol{\Sigma}$ upon observing that it plays no role in the Euclidean case (30). Therefore, the cost function for mean prediction Δ_m now becomes

$$\Delta_m = \mathtt{Log}_{\boldsymbol{\mu}}(\boldsymbol{\mu}_n)^\top\mathtt{Log}_{\boldsymbol{\mu}}(\boldsymbol{\mu}_n) \approx \mathtt{dist}^2(\boldsymbol{\mu}_n,\boldsymbol{\mu}), \quad (53)$$

where $\mathtt{dist}(\cdot,\cdot)$ denotes the geodesic distance between two manifold points. Finally, our estimator for mean prediction is given by

$$\widehat{\mathbf{s}}_m(\mathbf{x}) = \underset{\boldsymbol{\mu}\in\mathcal{M}}{\mathtt{argmin}}\sum_{n=1}^{N}\boldsymbol{\alpha}_n(\mathbf{x})\,\mathtt{dist}^2(\boldsymbol{\mu}_n,\boldsymbol{\mu}). \quad (54)$$

We note that the loss function used in (54) is the squared geodesic distance, which is SELF as shown by Rudi et al. (2018). To perform the estimation at a new test point $\mathbf{x}$, geometric optimization is required. In particular, we consider the Riemannian gradient descent, which extends the usual gradient descent method to manifolds with the guarantee that the computed value is still an element of the manifold.

By denoting the minimization objective of (54) as $\mathtt{F}(\boldsymbol{\mu})$, the iterative optimization process takes the form

$$\boldsymbol{\mu}_{i+1} = \mathtt{Exp}_{\boldsymbol{\mu}_i}\left(\eta_i\nabla_{\mathcal{M}}\mathtt{F}(\boldsymbol{\mu}_i)\right), \quad (55)$$

Algorithm 3: Imitation with Manifold Output

```
Collect multiple Riemannian trajectories;
Process raw data for D_M = {x_n, (μ_n, Σ_n)}_{n=1}^N;
Eigen-decomposition of covariance as per (59);
Initialize kernel k, regularization λ, and step size η;
for x = x_Start, ..., x_End do
    Input: a query point x;
    Calculate the weights α(x);
    repeat
        υ = ∇_M Σ_{n=1}^N α_n(x) dist²(μ_n, μ);
        μ ← R_μ(η υ);
    until convergence;
    Output: mean μ;
    Compute the parallel transport covariance Σ_{‖n} as
      per (58);
    Output: covariance Σ as per (56);
```

where $\nabla_{\mathcal{M}}$ is the Riemannian gradient operator and $\eta_i\in\mathbb{R}$ is a step size. Since the exponential map $\mathtt{Exp}$ could be difficult to compute, a computationally cheaper alternative is the retraction map $R_{\boldsymbol{\mu}}:\mathcal{T}_{\boldsymbol{\mu}}\mathcal{M}\to\mathcal{M}$, which is a first-order approximation to the exponential map. In addition to faster computation, the retraction map also guarantees convergence.

Apart from the Riemannian gradient descent, in some simple cases it is also possible to perform Riemannian optimization by optimizing in the Euclidean space first and then projecting the results onto the manifold. However, for complex manifolds, the projection operation can be very expensive to compute. By contrast, our employed Riemannian gradient descent provides a principled way for Riemannian optimization by making use of the intrinsic geometry of manifolds.

The procedure for the covariance matrix prediction can be achieved similarly to (31):

$$\boldsymbol{\Sigma} = \frac{\sum_{n=1}^{N}\boldsymbol{\alpha}_n(\mathbf{x})\left(\mathtt{Log}_{\boldsymbol{\mu}}(\boldsymbol{\mu}_n)\mathtt{Log}_{\boldsymbol{\mu}}(\boldsymbol{\mu}_n)^\top + \boldsymbol{\Sigma}_{\|n}\right)}{\sum_{n=1}^{N}\boldsymbol{\alpha}_n(\mathbf{x})}, \quad (56)$$

$$\approx \frac{\sum_{n=1}^{N}\boldsymbol{\alpha}_n(\mathbf{x})\boldsymbol{\Sigma}_{\|n}}{\sum_{n=1}^{N}\boldsymbol{\alpha}_n(\mathbf{x})}, \quad (57)$$

where a parallel transported covariance matrix $\boldsymbol{\Sigma}_{\|n}$ is used in place of $\boldsymbol{\Sigma}_n$. Parallel transport is necessary here as it can transfer the information from one point to another by considering the rotation of the coordinate systems along the geodesic curve. The transported covariance matrix can be computed as

$$\boldsymbol{\Sigma}_{\|n} = \sum_{j=1}^{d}\Gamma_{\boldsymbol{\mu}_n\to\boldsymbol{\mu}}(\mathtt{u}_j)\Gamma_{\boldsymbol{\mu}_n\to\boldsymbol{\mu}}(\mathtt{u}_j)^\top, \quad (58)$$

where $\mathtt{u}_j$ is obtained through an eigendecomposition:

$$\boldsymbol{\Sigma}_n = \sum_{j=1}^{d}\mathtt{u}_j\mathtt{u}_j^\top. \quad (59)$$

[†]Basic notions and nomenclatures on Riemannian manifolds are recalled in Appendix B.

Table 4. List of operations of common Riemannian manifolds considered in this paper.

	Sphere with radius r: $\mathbb{S}^2(r)$	Circular generalized cylinder: $\mathbb{R}^2 \times \mathbb{S}^1$
Distance metric $\texttt{dist}(\boldsymbol{\mu}_n, \boldsymbol{\mu})$	$r \arccos\left(\frac{\boldsymbol{\mu}_n^\top \boldsymbol{\mu}}{r^2}\right)$	$\sqrt{\|\boldsymbol{\mu}_n^{\mathbb{R}} - \boldsymbol{\mu}^{\mathbb{R}}\|^2 + \arccos\left(\boldsymbol{\mu}_n^{\mathcal{S}\top} \boldsymbol{\mu}^{\mathcal{S}}\right)^2}$
Minimization objective $\texttt{F}(\boldsymbol{\mu})$	$\sum_{n=1}^N \boldsymbol{\alpha}_n r^2 \arccos\left(\frac{\boldsymbol{\mu}_n^\top \boldsymbol{\mu}}{r^2}\right)^2$	$\sum_{n=1}^N \boldsymbol{\alpha}_n \left(\|\boldsymbol{\mu}_n^{\mathbb{R}} - \boldsymbol{\mu}^{\mathbb{R}}\|^2 + \arccos\left(\boldsymbol{\mu}_n^{\mathcal{S}\top} \boldsymbol{\mu}^{\mathcal{S}}\right)^2\right)$
Riemannian gradient $\nabla_{\mathcal{M}} \texttt{F}(\boldsymbol{\mu})$	$2\sum_{n=1}^N \boldsymbol{\alpha}_n \left(\frac{\boldsymbol{\mu}\boldsymbol{\mu}^\top}{r^2} - \mathbf{I}\right) \frac{\arccos\left(\frac{\boldsymbol{\mu}_n^\top \boldsymbol{\mu}}{r^2}\right)}{\sqrt{1 - \left(\frac{\boldsymbol{\mu}_n^\top \boldsymbol{\mu}}{r^2}\right)^2}} \boldsymbol{\mu}_n$	$2\sum_{n=1}^N \boldsymbol{\alpha}_n \left[\left[\boldsymbol{\mu}^{\mathbb{R}} - \boldsymbol{\mu}_n^{\mathbb{R}}\right]^\top \frac{\arccos\left(\boldsymbol{\mu}_n^{\mathcal{S}\top} \boldsymbol{\mu}^{\mathcal{S}}\right)}{\sqrt{1 - \left(\boldsymbol{\mu}_n^{\mathcal{S}\top} \boldsymbol{\mu}^{\mathcal{S}}\right)^2}} \times \left[(\boldsymbol{\mu}^{\mathcal{S}} \boldsymbol{\mu}^{\mathcal{S}\top} - \mathbf{I}) \boldsymbol{\mu}_n^{\mathcal{S}}\right]^\top \right]^\top$
Retraction $R_{\boldsymbol{\mu}}(\mathfrak{p})$	$r \frac{\boldsymbol{\mu} + \mathfrak{p}}{\|\boldsymbol{\mu} + \mathfrak{p}\|}$	$\left[\left[\boldsymbol{\mu}^{\mathbb{R}} + \mathfrak{p}^{\mathbb{R}}\right]^\top \left[\frac{\boldsymbol{\mu}^{\mathcal{S}} + \mathfrak{p}^{\mathcal{S}}}{\|\boldsymbol{\mu}^{\mathcal{S}} + \mathfrak{p}^{\mathcal{S}}\|}\right]^\top \right]^\top$
Logarithmic map $\texttt{Log}_{\boldsymbol{\mu}_n}(\boldsymbol{\mu})$	$\texttt{dist}(\boldsymbol{\mu}_n, \boldsymbol{\mu}) \frac{r^2 \boldsymbol{\mu} - \boldsymbol{\mu}_n^\top \boldsymbol{\mu} \boldsymbol{\mu}_n}{\|r^2 \boldsymbol{\mu} - \boldsymbol{\mu}_n^\top \boldsymbol{\mu} \boldsymbol{\mu}_n\|}$	$\left[\left[\boldsymbol{\mu}^{\mathbb{R}} - \boldsymbol{\mu}_n^{\mathbb{R}}\right]^\top \texttt{dist}(\boldsymbol{\mu}_n^{\mathcal{S}}, \boldsymbol{\mu}^{\mathcal{S}}) \frac{\boldsymbol{\mu}^{\mathcal{S}\top} - \boldsymbol{\mu}_n^\top \boldsymbol{\mu} \boldsymbol{\mu}_n^{\mathcal{S}\top}}{\|\boldsymbol{\mu}^{\mathcal{S}} - \boldsymbol{\mu}_n^\top \boldsymbol{\mu} \boldsymbol{\mu}_n^{\mathcal{S}}\|} \right]^\top$
Parallel transport $\Gamma_{\boldsymbol{\mu}_n \to \boldsymbol{\mu}}(\mathrm{u})$	$\mathrm{u} - \frac{\texttt{Log}_{\boldsymbol{\mu}_n}(\boldsymbol{\mu})^\top \mathrm{u}}{\texttt{dist}^2(\boldsymbol{\mu}_n, \boldsymbol{\mu})} \left(\texttt{Log}_{\boldsymbol{\mu}_n} \boldsymbol{\mu} + \texttt{Log}_{\boldsymbol{\mu}} \boldsymbol{\mu}_n\right)$	$\left[\mathrm{u}^{\mathbb{R}\top} \; \mathrm{u}^{\mathcal{S}\top} - \frac{\texttt{Log}_{\boldsymbol{\mu}_n^{\mathcal{S}}}(\boldsymbol{\mu}^{\mathcal{S}})^\top \mathrm{u}^{\mathcal{S}}}{\texttt{dist}^2(\boldsymbol{\mu}_n^{\mathcal{S}}, \boldsymbol{\mu}^{\mathcal{S}})} \left[\texttt{Log}_{\boldsymbol{\mu}_n^{\mathcal{S}}} \boldsymbol{\mu}^{\mathcal{S}} + \texttt{Log}_{\boldsymbol{\mu}^{\mathcal{S}}} \boldsymbol{\mu}_n^{\mathcal{S}}\right]^\top \right]^\top$

Similarly to the Euclidean setting, manifold-adaptive behavior, such as passing through some desired via-point, is realized by modifying the weight $\boldsymbol{\alpha}$ according to (19).

Table 4 provides the manifold operations required for two problems of interest considered in this paper: The sphere $\mathbb{S}^2(r)$ with radius r and the circular generalized cylinder $\mathbb{R}^2 \times \mathbb{S}$. An algorithm for imitation with manifold-valued output is summarized in Algorithm 3.

4 Experimental Evaluations

In this section, we evaluate the effectiveness of our proposed approach with both simulations and real experiments. Specifically, we conduct simulations on trajectory reproductions in Section 4.1, where the resulting imitation fidelity is reported. We then test the performance of our algorithm in the case of adaptive behavior including both position and velocity via-points in Section 4.2. For manifold trajectory learning, we study the problem of reproducing and adapting trajectory on a sphere and a generalized cylinder as in Section 4.3. Finally, in Section 4.4 we report the experimental results on a real KUKA robot arm. A video of the real-world experiments is available in the supplementary material.

4.1 Trajectory Reproduction

We first evaluate the effectiveness of our algorithm for trajectory reproduction. Our goal is to compare with state-of-the-art algorithms in terms of imitation fidelity by learning the trajectories of four illustrative letters, namely 'N', 'S', 'W', and 'P', with time as the input. It should be noted that trajectory covariance learning is usually not considered by autonomous approaches like LPV-DS as they focus more on global stability and robustness to disturbances.

The collected data from multiple demonstrations for the letter trajectories are first processed with Gaussian mixture models (GMM). Subsequently, a probabilistic reference trajectory is extracted by Gaussian mixture regression (GMR) (Billard et al. 2008), serving as the baseline for different imitation algorithms to compare against.

The learning results are shown in Figure 3. In the same row, the same multiple demonstrations for one letter are fed to different imitation learning algorithms in order to learn the mean and the covariance of a probabilistic trajectory. The first column in yellow represents ProMP. ProMP is based on a parametric method, so a set of basis functions is needed to fit the demonstration trajectories. We employ 30 Gaussian radial basis functions (RBF) for learning reference trajectories. The second column in blue represents KMP, whose hyperparameters, such as the regularization coefficients, are selected according to Huang et al. (2019). The third column in red denotes LPV-DS. In order for LPV-DS to generate a trajectory, an initial point needs to be manually selected. Here, we set it to coincide with the reference trajectory's initial point. The last two columns in green represent our approach. Wherein, the fourth column denotes the KL divergence imitation mode and the fifth column denotes the reverse KL divergence imitation mode. To apply our algorithm, we choose the Gaussian kernel, which is defined by

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\kappa \parallel \mathbf{x}_i - \mathbf{x}_j \parallel^2) \tag{60}$$

where we set hyper-parameter $\kappa = 6$ when training the input-dependent weights $\boldsymbol{\alpha}$.

In general, all imitation learning algorithms can correctly reproduce the originally demonstrated trajectories. Both learned trajectory mean and covariance coincide with the reference probabilistic trajectory to some extent. In Figure 3, it can be observed that our proposed methods most closely reproduce the original demonstrations. Especially, when it comes to abrupt turns, as in the case of the letter 'W', our algorithm can still imitate the demonstrated trajectory closely, while other methods exhibit a larger mismatch.

To quantitatively compare the reproduction performances of each algorithm, we report the cumulative error for

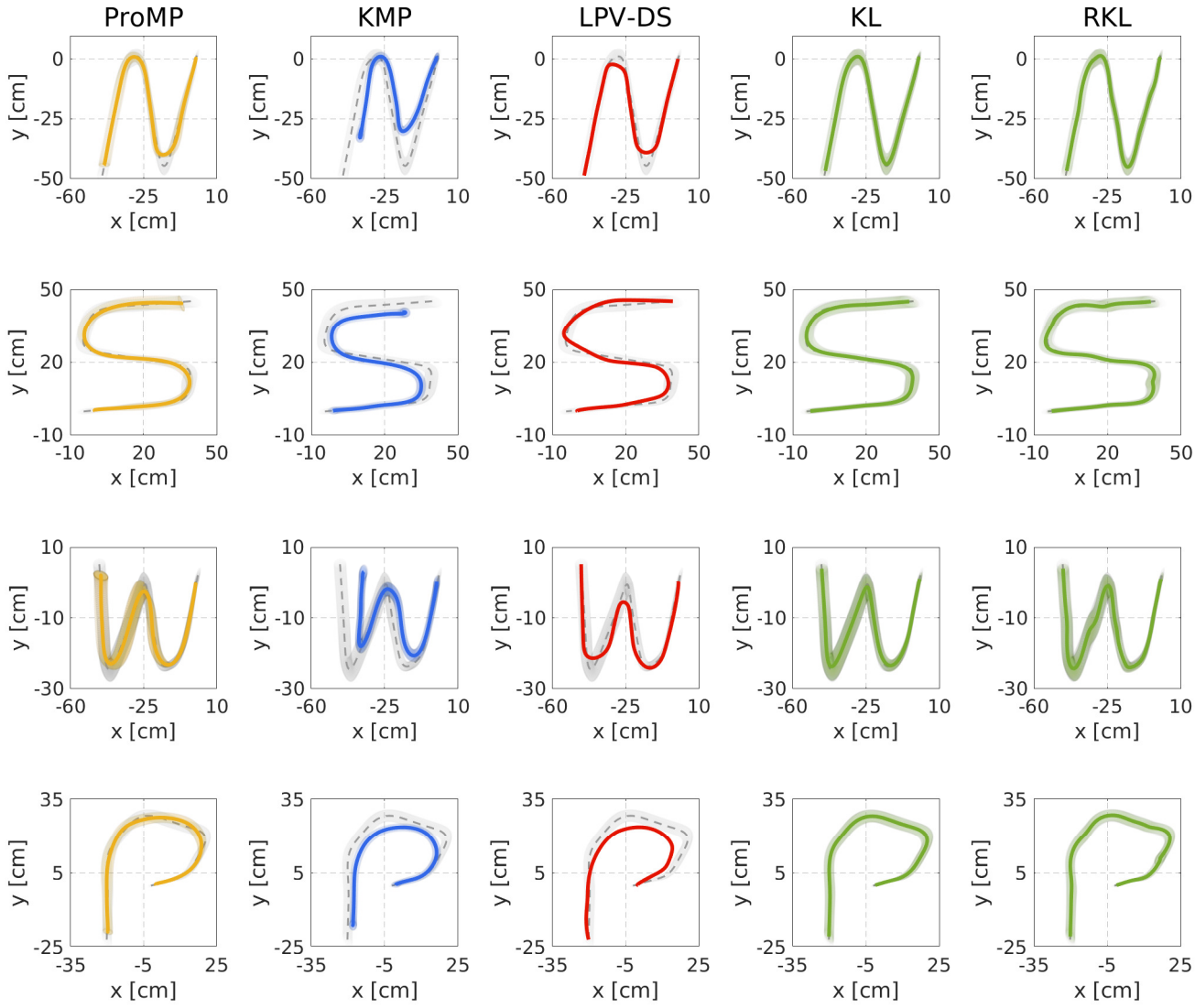


Figure 3. Imitation fidelity comparison among ProMP (yellow), KMP (blue), LPV-DS (red) and our approach (green, with the fourth column KL imitation mode and the fifth column reverse KL imitation mode) for reproducing the trajectories of different letters. Solid curves and shallow areas are used to represent trajectory mean and standard deviation, respectively. The reference baseline used here is the probabilistic trajectory retrieved by GMR (gray).

Table 5. Performance comparison of different imitation learning algorithms

	'N'			'S'			'W'			'P'		
	C_m	C_{cov}	Time (s)	C_m	C_{cov}	Time (s)	C_m	C_{cov}	Time (s)	C_m	C_{cov}	Time (s)
ProMP	17.5	16.5	0.01	15.8	19.4	0.02	14.2	17.7	0.02	14.6	14.1	0.01
KMP	40.3	19.3	1.84	30.6	17.5	1.83	35.2	18.4	1.88	29.2	18.8	2.10
LPV-DS	60.8	N/A	12.9	65.3	N/A	11.1	61.0	N/A	13.5	56.9	N/A	9.5
Ours (KL)	7.4	5.7	0.11	8.9	6.5	0.09	7.8	6.0	0.11	5.9	5.4	0.09
Ours (RKL)	7.3	6.4	1.47	9.3	5.7	1.51	8.9	6.2	1.55	5.0	5.1	1.52

trajectory mean and covariance, as well as the training time. For evaluating the trajectory mean imitation, we choose the Root Mean Square Error (RMSE) between the estimated value and the demonstrated one: $C_m = \sqrt{\sum_{n=1}^{N} \parallel \widehat{\mathbf{s}}_m(\mathbf{x}_n) - \boldsymbol{\mu}_n \parallel^2}$. Likewise, for the evaluation of trajectory covariance imitation, we choose the error as $C_{cov} = \sqrt{\sum_{n=1}^{N} \parallel \log(\boldsymbol{\Sigma}_n^{-\frac{1}{2}} \widehat{\mathbf{s}}_c(\mathbf{x}_n) \boldsymbol{\Sigma}_n^{-\frac{1}{2}}) \parallel_{\mathrm{F}}^2}$, where the notation of geodesic distance of positive definite matrices is utilized. Concerning computational complexity, ProMP is expected to be more efficient since it grows as $\mathcal{O}(m^3 d^3)$, where m is the dimension of the basis function and d is the dimension of output value. KMP and our algorithm both have a $\mathcal{O}(n^3)$ time complexity, where n denotes the number of trajectory data points. LPV-DS is expected to be the most computationally expensive since a non-linear

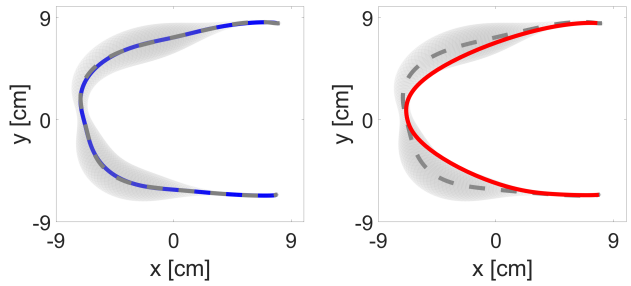


Figure 4. Comparison of imitation performances by the KL (blue) and RKL (red) imitation modes, where the dashed lines represent the reference mean and the shallow areas represent the trajectory covariance.

constrained optimization problem has to be solved. The obtained numerical results are summarized in Table 5, where training time is averaged over five trials. To conclude, our algorithm shows significant performance improvements with respect to the selected state-of-the-art methods on a diverse set of target trajectories, in terms of both trajectory mean and covariance imitation quality.

In addition, we would like to further clarify the difference in the imitation behaviors exhibited by the KL and RKL imitation modes. In particular, we present the individual performance in terms of mean prediction given a C-shape probabilistic trajectory, as shown in Figure 4. It can be observed that when imitating in the KL mode, the reproduced trajectory closely follows the demonstrated trajectory mean as it is only dependent on the demonstrated trajectory mean. On the contrary, the reproduced trajectory using the RKL mode is not only dependent on the demonstrated trajectory mean but also on the demonstrated trajectory covariance, as evidenced by the larger deviation in the region with larger covariance. Different imitation modes will allow for more flexibility in probabilistic imitation learning. For example, trajectory covariance can be exploited to prioritize trajectory mean prediction to balance task importance.

4.2 Trajectory Adaptation

In this experiment, we study the problem of trajectory adaptation considering two scenarios: Position only and both position and velocity. In position adaptation, three cases are studied, namely start-point, mid-way-point, and end-point adaptation. Similarly to Section 4.1, ProMP and KMP are employed for comparison purposes. LPV-DS is not considered here as it does not provide a straightforward manner for trajectory adaptation. We use the KL divergence imitation mode for our approach.

For position adaptation, we consider modulating a C-shape trajectory to go through variously positioned via-points with time as the input. We first evaluate start-point adaptation. We set a desired point at $[5\ 6]^\top \mathrm{cm}$ at time step $t = 0\,\mathrm{s}$. Next, we move the desired point to $[-5\ 0]^\top \mathrm{cm}$ at time step $t = 0.5\,\mathrm{s}$ to study mid-way-point adaptation. Finally, the issue of end-point adaptation is studied by setting the desired point at $[10\ -9]^\top \mathrm{cm}$ at time step $t = 1.0\,\mathrm{s}$. Different trajectory position adaptation cases are illustrated in Figure 5. It shall be observed that all algorithms are capable of respecting the additional via-point constraint with high accuracy while the trajectory generated by our

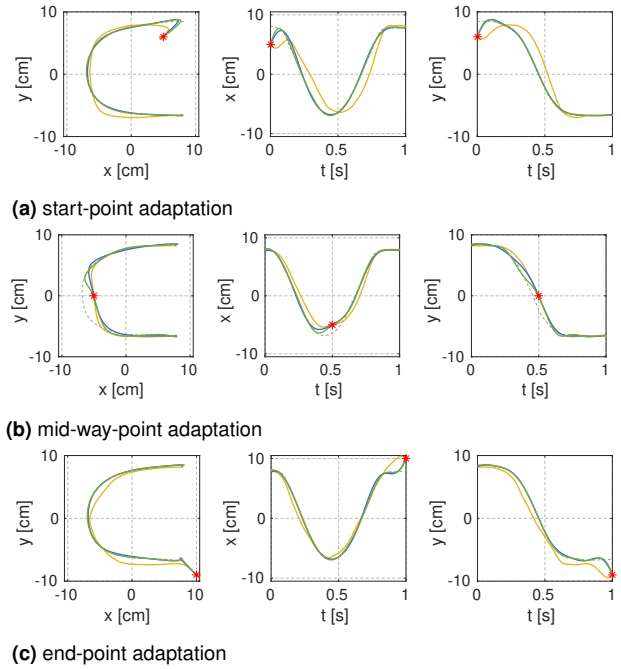


Figure 5. Comparison of position adaptation among ProMP (yellow), KMP (blue), and our approach (green) for a C-shape trajectory with time as the input. The desired points to go through are marked with red stars.

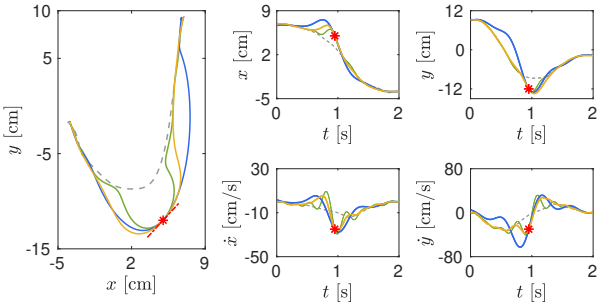
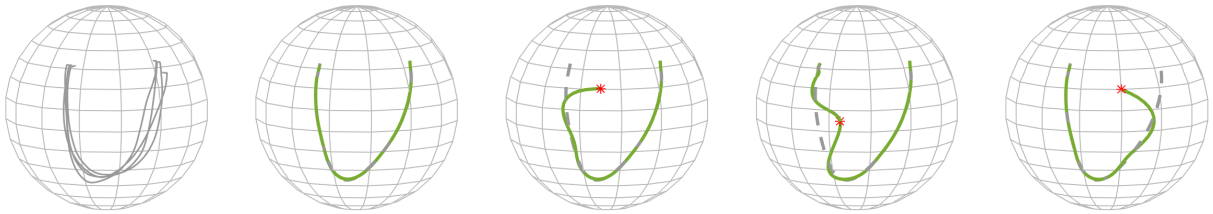


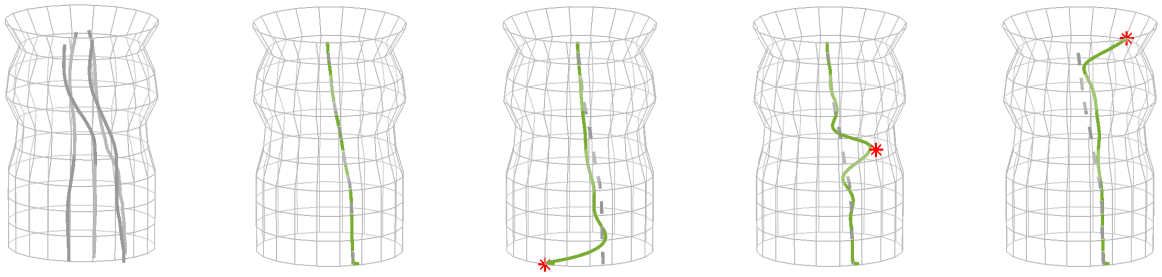
Figure 6. Comparison of both position and velocity adaptation among ProMP (yellow), KMP (blue), and our approach (green) for a J-shape trajectory (dashed gray) with time as the input. The desired position points to go through are marked with red stars and the desired velocity is depicted with a dashed red line.

algorithm tends to converge to the original demonstrated trajectory closer compared with other methods.

In the next experiment, we study the performance of adapting both the position and the velocity of an imitated trajectory. We consider modulating a J-shape trajectory that has a total duration of $2\,\mathrm{s}$. For ProMP, additional first-order derivatives of basis functions are needed to encode the dynamics of the motion. By contrast, non-parametric methods like KMP and our approach do not require basis functions. To adapt the trajectory, at time step $t = 1\,\mathrm{s}$ we set a desired location at $[5\ \ -12]^\top \mathrm{cm}$ associated with a desired velocity $[-25\ \ -30]^\top \mathrm{cm/s}$. The obtained results are shown in Figure 6, where all the algorithms successfully generate trajectories that can pass through the desired position and desired velocity. The adaptation errors at the via point are negligible. It can be observed that the trajectory generated by our algorithm is affected the least by the additional



(a) Trajectory imitation on a sphere



(b) Trajectory imitation on a generalized cylinder

Figure 7. Illustration of trajectory reproduction and adaptation on a manifold where multiple demonstrations are depicted in gray, trajectories generated by our algorithm are plotted in green and the desired via-point is marked with a red star.

adaptation requirements, while the other algorithms end up with larger imitation errors due to larger deviations.

4.3 Manifold Trajectory Learning

In this experiment, we demonstrate a unique feature of our approach, namely, learning and adapting trajectories lying on manifolds. To show the effectiveness of our approach, we study the problem of imitation on two common manifolds, namely a sphere ($\mathbb{S}^2$) and a circular generalized cylinder ($\mathbb{R}^2 \times \mathbb{S}$). The performance of both reproduction from multiple demonstrations as well as the adaptation towards a via point is evaluated. Since adapting trajectories on manifolds has rarely been addressed before in robot imitation learning, we hereby only report the experimental results of our approach.

For learning on a sphere, we first draw four U-shaped trajectories on a sphere, each having time duration $t = 1\,\mathrm{s}$, as shown in the first column of Figure 7a. The radius of the sphere is $1\,\mathrm{cm}$ and the center is located at the origin of the coordinate system. The base point for conducting exponential and logarithmic mappings is chosen as $[0\ 0\ 1]^\top \mathrm{cm}$. Likewise, the collected demonstration data is processed using GMM and a reference probabilistic trajectory is subsequently extracted by GMR. As the collected data is defined on a Riemannian manifold, therefore, extended versions of GMM and GMR for Riemannian manifolds are employed for data processing (Zeestraten et al. 2017b).

To show the adaptability on the manifold, a via-point at $[-0.199\ \ -0.98\ 0]^\top \mathrm{cm}$ is set on the surface of the sphere for the trajectory to pass through at $t_v = 0.35\,\mathrm{s}$. The second and third columns of Figure 7a show reproduction and adaptation trajectories from our algorithm, respectively. During each point prediction, the step size of Algorithm 3 is set to $\eta = 0.01$. We can observe that our algorithm is capable of accurately meeting the requirement of via-point (marked with a red star) adaptation while preserving the shape of the original demonstrated trajectory.

The other experiment we carry out is trajectory imitation on a circular generalized cylinder, which has a smoothly varying circular cross-section. We use the cylindrical coordinate system to express a data point (r, z, φ), where r is the radial distance from the z-axis, z denotes height, and φ is the azimuth. Since a data point lying on a generalized cylinder consists of a two-dimensional Euclidean component and a circular component, we formulate its Riemannian representation with the help of the Cartesian product: $\mathbb{R}^2 \times \mathbb{S}^1$. The center of the bottom of the cylinder is positioned at the origin of the cylindrical coordinate system and the cylinder center line is aligned with the z-axis.

We first draw four demonstrations along the cylinder's central line, each one of $1\,\mathrm{cm}$ length and duration $t = 1\,\mathrm{s}$, as shown in the first column of Figure 7b. The second column of Figure 7b compares the reference retrieved by GMR and reproduction trajectory by our approach with step size again set to $\eta = 0.01$. It can be observed that our algorithm is able to reproduce the reference trajectories with high imitation fidelity. At time step $t_v = 0.5\,\mathrm{s}$, the trajectory is required to adapt towards a via-point located at $(1.51\,\mathrm{cm}, 0.5\,\mathrm{cm}, 0.863\,\mathrm{rad})$. The learning results are shown in the third column of Figure 7b, where the adapted trajectory exactly passes through the via-point (marked with a red star).

4.4 Polishing Task

The real-world experiment for evaluation of the proposed method is conducted via a polishing task. The purpose of the polishing task is to teach a robot by kinesthetic guidance so that it learns how to polish on the surface of a sphere manifold. Once reproduction of the skill is achieved, the robot is later required to polish a user-defined via-point

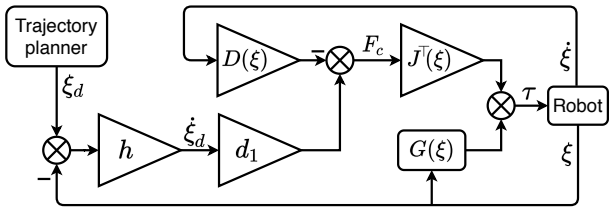


Figure 8. Block illustration of the employed passive controller. The algorithm is composed of a velocity-based control law and a gravity-compensation term.

on the surface to exhibit adaptive behavior enabled by our approach.

4.4.1 Experimental setup The robot used for accomplishing the polishing task is the KUKA LWR IV+, a 7-DoF robotic arm. With the joint torque sensors mounted at the actuators, it can be torque-controlled and the torque commands are sent to KUKA using the Fast Research Interface (FRI) at 500 Hz. There is also a 6-axis force-torque sensor installed on the end-effector. We choose to attach a 3D-printed finger tool to the robot as our end-effector. In addition, a burst-resistant exercise ball is used as the sphere that the robot will polish on.

To transfer polishing skills from the human to the robot, we demonstrate the same task four times. During the demonstration, the robot is switched to the gravity compensation mode to make the robot light to be interacted with. The Cartesian trajectory of the robot end-effector is recorded to train our algorithm.

4.4.2 Robot control Tracking time-indexed position profiles usually poses a challenge for robots to achieve fast reactivity in response to external disturbances. To this end, we employ a passive controller developed by Kronander and Billard (2015) to enhance the robot's robustness to large real-time disturbances. We consider the control of the robot's end effector in Cartesian space as in (Amanhoud et al. 2019); the corresponding translational control force $\mathbf{F}_c$ takes the form

$$\mathbf{F}_c = \mathbf{D}(\boldsymbol{\xi})(\dot{\boldsymbol{\xi}}_d - \dot{\boldsymbol{\xi}}) = d_1\dot{\boldsymbol{\xi}}_d - \mathbf{D}(\boldsymbol{\xi})\dot{\boldsymbol{\xi}}, \tag{61}$$

where $\boldsymbol{\xi}$ represents the end-effector position and $\mathbf{D}(\boldsymbol{\xi})$ is a state-varying damping matrix with the first eigenvector $d_1 > 0$ aligned with the desired velocity $\dot{\boldsymbol{\xi}}_d$. For our time-invariant task representation, we design the desired velocity profile as

$$\dot{\boldsymbol{\xi}}_d = h(\boldsymbol{\xi}_d - \boldsymbol{\xi}), \tag{62}$$

where $h > 0$ weighs the tracking deviation.

In our experiment, the passive controller is only applied to the translational direction. For orientation control, a control moment is computed by setting the end-effector roughly pointing towards the center of the ball using spherical linear interpolation with a PD control law. Finally, the control commands are computed by mapping the control wrench, consisting of control moment and force, to joint torques using the corresponding robot Jacobian matrix. Figure 8 presents a block representation of our proposed controller.

4.4.3 Results and analysis An illustration of the kinesthetic demonstration procedure as well as skill reproduction is shown in Figure 9. A teacher demonstrates the polishing

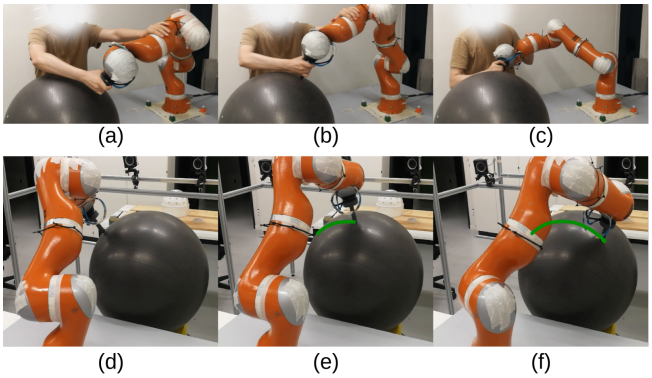


Figure 9. Snapshots of demonstration (*top row*) and reproduction (*bottom row*) of the polishing task with KUKA LWR IV+ robot arm.

task multiple times to the robot on a sphere whose radius is 0.3m and the center position is identified as $[-0.7631\ -0.0271\ 0.0698]^\top$m in a least-squares sense. We then evaluate the reproduction capabilities of our approach on the robot. It shall be observed that the robot is able to reproduce the demonstrated task along the surface of the sphere.

In another reproduction experiment, we apply external perturbations to the robot to test its working robustness. By lifting the robot arm, the robot is still able to recover the originally planned trajectory. The satisfactory compliant behavior makes it safe for the robot to interact with a human user. To evaluate the adaptation capability of our approach, we set different via points for the robot end-effector to pass through. Figure 10 shows that the robot end-effector movement is modulated to respect the constraint incurred by different additional desired via-points. The real robot trajectories are plotted in Figure 11, where the relative position between the trajectories and the ball as well as the zoomed-in trajectories are provided for clarity. Reproduction and adaptation performances are also reported in the supplementary video.

Though, in theory, the trajectory generated by our approach should exactly lie on the manifold, sometimes there is small interference or deviation observed between the end-effector and the ball. This could be a result of the measuring error of the sphere information as well as the tracking error between the sent reference trajectory and the real robot end-effector position. In the future, this issue could be resolved by introducing contact perception signals at the level of the control architecture design.

5 Discussions

In this section, we first cover and compare with previous related work and analyze the connections (Section 5.1). Then, we consider possible extensions to complement our proposed method and envisage future works (Section 5.2).

5.1 Related Work and Connections

The algorithmic foundations of robot imitation learning notably include regression techniques (see Table 1) such as linear regression with basis functions (e.g., DMP and ProMP), kernel ridge regression (e.g., KMP), Gaussian mixture regression (e.g., SEDS), and Gaussian process

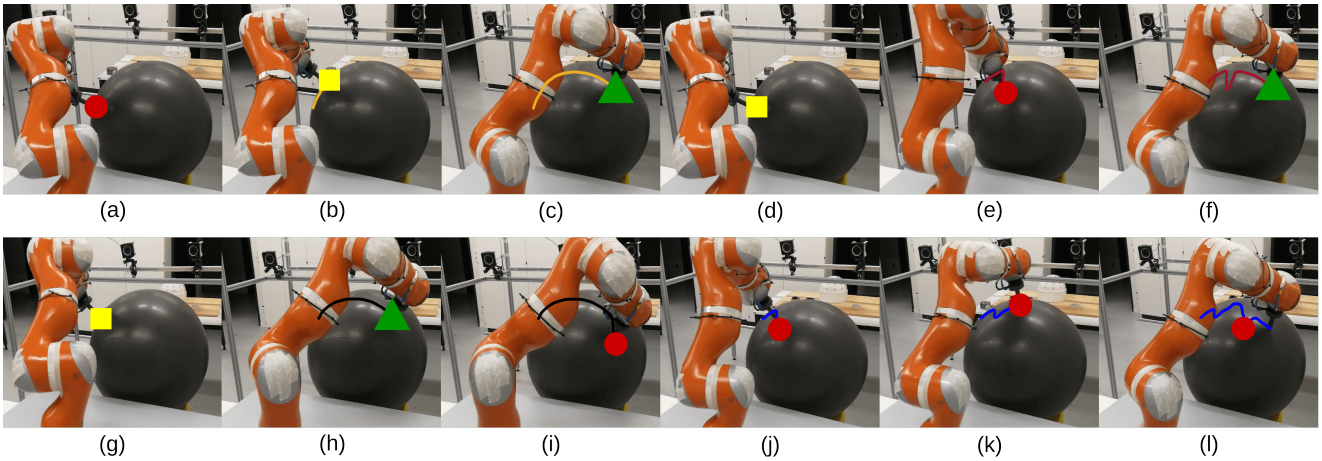


Figure 10. Snapshots of the trajectory adaptation on a sphere with KUKA LWR IV+, in terms of a new desired start point((a)-(c)), a new mid-way point ((d)-(f)), a new desired endpoint ((g)-(i)) as well as multiple via-points((j)-(l)). The start points and the end points of the original demonstration trajectories and new desired points are marked with squares, triangles, and circles.

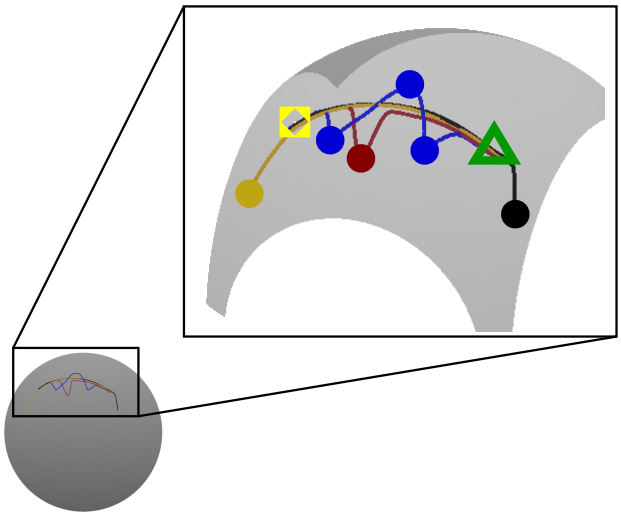


Figure 11. Plot of the robot end-effector adaptive trajectories in the task of polishing on the sphere, where the yellow, red, black, and blue curves represent start-point, mid-way-point, end-point, and multi-via-point adaptation, respectively.

regression (e.g., LGP and GPM), just to name a few. Although it is possible to bypass the regression step by calculating the arithmetic mean as done by Learning from demonstration by Averaging Trajectories (LAT), the practical functionality of LAT can be severely limited due to the lack of trajectory adaptation.

Our method also follows the supervised learning paradigm. Remarkably, by leveraging structured prediction it achieves two main goals: The capability of prediction in structured output spaces and the flexibility of selecting different loss functions.

It is essential to investigate the most relevant types of output data for robot imitation learning, since their correct processing is crucial for successful task execution. Such output spaces usually possesses some geometric structure (Calinon 2020), e.g., when transferring skills from human experts to robots. Compared with most movement primitives approaches that capture motion patterns by processing demonstrated data under the Euclidean metric, our approach distinguishes itself by the capability of prediction in structured output spaces such as Riemannian manifolds, which recently attracted growing interest in robotics (Zeestraten et al. 2017b; Beik-Mohammadi et al. 2021). Besides manifold-structured data, our framework preserves the potential to handle also other output types (e.g., histograms, graphs, time series, etc.) thanks to the generality of the structured prediction paradigm.

It is also important to determine the similarity metric between the expert's and the learner's behavior. Given the very same demonstrated dataset, different loss functions usually result in different imitation policies (Duan et al. 2020). Unlike conventional movement primitives that quadratically penalize deviation from the demonstrated trajectory, our approach permits a wide spectrum of loss functions thanks to the adopted implicit encoding framework for structured prediction. To construct a proper loss function for probabilistic trajectory imitation, we take the novel point of view that imitation learning in sequential decision-making can be viewed as f-divergence minimization between learner and expert policy, as indicated by recent studies (Ke et al. 2021; Ghasemipour et al. 2020).

Being a common measure of the difference between two probability distributions, f-divergence also emerges as a popular tool in other robot learning settings. In reinforcement learning algorithms such as Relative Entropy Policy Search (REPS) (Peters et al. 2010) and Trust Region Policy Optimization (TRPO) (Schulman et al. 2015), a KL-divergence constraint is imposed between successive parameterized policies to prevent too large policy updates leading to unknown regions of the state space. Instead, in our approach the role of the f-divergence is to determine the coupling between the trajectory mean and covariance. For example, when using the KL imitation mode for trajectory mean prediction, trajectory covariance is not involved. By contrast, when using the RKL imitation mode for trajectory mean prediction, trajectory covariance is also employed. This can provide an interface to incorporate the relative importance of demonstrations, i.e., small covariance represents high importance and will weigh more in trajectory generation (Huang et al. 2019). The different imitation

modes as a result of different f-divergence functions thus enable multiple coupling modalities between trajectory mean and covariance in the case of probabilistic trajectory imitation.

Another related field is imitation for sequential decision-making, where a learner makes decisions by mimicking an expert given the demonstrated dataset of state-action pairs. Notably, one of the fundamental issues in behavior cloning are the so-called compounding errors as formalized by DAgger (Ross et al. 2011). To alleviate the issue, DAgger iteratively appends training data to the dataset, yielding an interactive procedure for imitation. Focusing on the form of the policy, implicit behavior cloning (IBC) casts imitation as a conditional energy-based modeling problem by using implicit models, leading to improvement in visuomotor tasks (Florence et al. 2022). Aiming at providing a unifying framework for imitation, a game-theoretic perspective to imitation is proposed in (Swamy et al. 2021) to minimize the worst-case divergence, and performance bounds for each imitation setting are analyzed. Though sharing the same spirit of imitation, our approach directly concentrates on imitating robot trajectories, which gives rise to trajectory generation programmed by expert demonstrations.

Distilling the underlying reward, which is typically assumed to be linear in the features, is also a notable paradigm for imitation. Based on the principle of maximum entropy, Maximum Entropy Inverse Reinforcement Learning (MaxEntIRL) recovers a reward by resolving the ambiguity in the matching of feature counts (Ziebart et al. 2008). Note that when using MaxEntIRL to generate robot trajectories a subsequent motion planning module is necessary (Ruan et al. 2023). Geometric trajectory generation requires dedicated motion planning methods, such as (Bonalli et al. 2019) and (Kingston et al. 2019), which might complicate the procedure compared to our approach.

5.2 Limitations and Future Work

For the sake of completeness, let us mention some current limitations in our proposed approach and potential future improvements. As a memory-based approach, our algorithm may encounter difficulties when dealing with large-scale datasets (like other kernel methods). Nevertheless, such a limitation may be alleviated by adopting sub-sampling techniques to reduce the kernel memory footprint (Rudi et al. 2015). Besides, the requirement of performing Riemannian gradient descent for geometric trajectory imitation prevents the usage of our method on the fly. Therefore, it may be beneficial to investigate more efficient Riemannian optimization strategies. In addition, when directly applying our method for making sequential decisions, it may suffer from compounding errors at test time.

For different imitation modes, we considered two representative f-divergence functions for the design of the loss function, namely the KL and the RKL divergences. With these two loss functions being SELF, it will be interesting to examine if other f-divergence functions also satisfy the SELF condition. At a first glance, it can be readily shown that the loss function is SELF when using the Jeffreys divergence given by $f(u) = (u-1)\log(u)$ (e.g., see (Pardo 2018)), which is composed of the summation of the KL and the RKL divergences: $D_{\mathtt{JD}}(\tilde{\mathbf{y}}_n, \tilde{\mathbf{y}}) = D_{\mathtt{KL}}(\tilde{\mathbf{y}}_n, \tilde{\mathbf{y}}) + D_{\mathtt{RKL}}(\tilde{\mathbf{y}}_n, \tilde{\mathbf{y}})$. For future work, we would like to investigate other types of f-divergence functions for the loss design. In terms of applications, we conceive that our approach could be applied in a variety of robotic applications that require human-like trajectory generation such as whole-body teleoperation for humanoid robots (Darvish et al. 2023).

6 Conclusions

In this paper, we present a novel robot imitation algorithm for probabilistic trajectory imitation. Specifically, an implicit embedding framework for structured prediction is employed, which endows our developed movement primitives with the capability of prediction with structured output space and the flexibility of choosing different loss functions. Specifically, the loss functions used in the structured prediction are constructed by leveraging a novel point of view for imitation learning, i.e., minimization of the f-divergence between an expert's and a learner's policy. By choosing different types of f-divergences, we are able to learn demonstrated policies with different imitation modes.

We illustrate the effectiveness of our proposed approach by comparing it with state-of-the-art methods. Our results demonstrate that our approach can increase imitation fidelity to a considerable extent in terms of both trajectory mean and covariance prediction while preserving merits such as multi-dimensional inputs and spatial and temporal trajectory modulations. Furthermore, our algorithm can be extended to learning and adapting trajectories on Riemannian manifolds, which distinguishes our approach from traditional methods.

Acknowledgements

This research was supported by the Swiss National Science Foundation through the National Center of Competence in Research (NCCR) Robotics. Part of this work has been carried out at the Machine Learning Genoa Center, Università di Genova, Italy. D. P. acknowledges the support by the An.Dy project, which has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No. 731540. L. R. acknowledges the financial support of the European Research Council (grant SLING 819789), the Center for Brains, Minds and Machines, funded by NSF STC award CCF-1231216, the AFOSR projects FA9550-18-1-7009, FA9550-17-1-0390 and BAA-AFRL-AFOSR-2016-0007 (European Office of Aerospace Research and Development), the EU H2020-MSCA-RISE project NoMADS - DLV-777826, and the NVIDIA Corporation for the donation of a Titan Xp GPUs and a Tesla k40 GPU. R. C. acknowledges the following: This study was carried out within the FAIR - Future Artificial Intelligence Research and received funding from the European Union Next-GenerationEU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR) – MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.3 – D.D. 1555 11/10/2022, PE00000013). This manuscript reflects only the authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them.

Appendix

A Proof Sketch for SELF

Here we provide sketch proof that the KL and the reverse KL divergence-based loss functions are SELF. Our proof is

done by construction, i.e., we show that the loss functions can be expressed in the form of (4). The central idea is to explicitly determine the feature map $\mathbf{c}$ and the linear operator V. For convenience, we show that the mean and covariance cost functions can be constituted in a SELF fashion, which can readily lead to the loss functions being SELF.

A.1 KL divergence Consider the following formulation for the cost function when making the mean prediction:

$$\begin{aligned}&(\boldsymbol{\mu}-\boldsymbol{\mu}_n)^\top\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}-\boldsymbol{\mu}_n)\\&=\underbrace{\begin{bmatrix}\boldsymbol{\mu}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\\ \boldsymbol{\mu}\\ 1\end{bmatrix}}_{\mathbf{c}(\boldsymbol{\mu})}{}^\top\underbrace{\mathtt{blkadiag}(1,-2\boldsymbol{\Sigma}^{-1},1)}_{V}\underbrace{\begin{bmatrix}\boldsymbol{\mu}_n^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_n\\ \boldsymbol{\mu}_n\\ 1\end{bmatrix}}_{\mathbf{c}(\boldsymbol{\mu}_n)}\end{aligned}\tag{63}$$

where $\mathtt{blkadiag}(\cdot)$ denotes the block anti-diagonal matrix.

Consider the following formulation for the cost function when making covariance prediction:

$$\begin{aligned}&(\boldsymbol{\mu}-\boldsymbol{\mu}_n)^\top\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}-\boldsymbol{\mu}_n)+\log|\boldsymbol{\Sigma}|+\mathtt{Tr}(\boldsymbol{\Sigma}^{-1}\boldsymbol{\Sigma}_n)\\&=\underbrace{\begin{bmatrix}\boldsymbol{\mu}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\\ \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\\ \boldsymbol{\phi}_2(\boldsymbol{\Sigma}^{-\frac{1}{2}})\\ \log|\boldsymbol{\Sigma}|\\ \mathtt{vec}(\boldsymbol{\Sigma})\\ 1\\ \boldsymbol{\phi}_1(\boldsymbol{\mu})\\ -2\boldsymbol{\mu}\\ 1\end{bmatrix}}_{\mathbf{c}(\boldsymbol{\Sigma})}{}^\top\underbrace{\left[\begin{array}{c:c:c}\mathbf{0}&\mathtt{blkadiag}&\\ &(1,\mathbf{I}_d,\mathbf{I}_q,&\\ &1,\mathbf{0}_{d^2},\mathbf{I}_{d^2})&\\ \hdashline\mathbf{0}&&\mathbf{0}\end{array}\right]}_{V}\underbrace{\begin{bmatrix}\boldsymbol{\mu}_n^\top\boldsymbol{\Sigma}_n^{-1}\boldsymbol{\mu}_n\\ \boldsymbol{\Sigma}_n^{-1}\boldsymbol{\mu}_n\\ \boldsymbol{\phi}_2(\boldsymbol{\Sigma}_n^{-\frac{1}{2}})\\ \log|\boldsymbol{\Sigma}_n|\\ \mathtt{vec}(\boldsymbol{\Sigma}_n)\\ 1\\ \boldsymbol{\phi}_1(\boldsymbol{\mu}_n)\\ -2\boldsymbol{\mu}_n\\ 1\end{bmatrix}}_{\mathbf{c}(\boldsymbol{\Sigma}_n)}\end{aligned}\tag{64}$$

where $\mathbf{0}$ is the zeros matrix with proper dimension. The features $\boldsymbol{\phi}_1(\cdot)$ and $\boldsymbol{\phi}_2(\cdot)$ are designed such that we have $\boldsymbol{\phi}_2(\boldsymbol{\Sigma}^{-\frac{1}{2}})^\top\boldsymbol{\phi}_1(\boldsymbol{\mu}_n)=\boldsymbol{\mu}_n\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_n$. Also, we have $\mathtt{vec}(\boldsymbol{\Sigma})=\left[\mathtt{vecr}(\boldsymbol{\Sigma})^\top\ \ \mathtt{vecr}(\boldsymbol{\Sigma}^{-1})^\top\ \ \mathtt{vecc}(\boldsymbol{\Sigma})^\top\ \ \mathtt{vecc}(\boldsymbol{\Sigma}^{-1})^\top\right]^\top$, where $\mathtt{vecr}(\cdot)$ and $\mathtt{vecc}(\cdot)$ denote the row- and column-major order vectorization operation for a matrix.

A.2 Reverse KL divergence Similarly, the cost functions used in the reverse KL divergence can be expressed in the form of SELF. For the cost function of mean prediction, consider the following formulation:

$$\begin{aligned}&(\boldsymbol{\mu}-\boldsymbol{\mu}_n)^\top\boldsymbol{\Sigma}_n^{-1}(\boldsymbol{\mu}-\boldsymbol{\mu}_n)\\&=\underbrace{\begin{bmatrix}\boldsymbol{\mu}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\\ -2\boldsymbol{\mu}\\ \boldsymbol{\phi}_1(\boldsymbol{\mu})\\ \boldsymbol{\phi}_2(\boldsymbol{\Sigma}^{-\frac{1}{2}})\\ \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\\ 1\end{bmatrix}}_{\mathbf{c}(\boldsymbol{\mu})}{}^\top\underbrace{\mathtt{blkadiag}\begin{pmatrix}0,\mathbf{I}_d,\mathbf{I}_q\\ \mathbf{0}_q,\mathbf{0}_d,1\end{pmatrix}}_{V}\underbrace{\begin{bmatrix}\boldsymbol{\mu}_n^\top\boldsymbol{\Sigma}_n^{-1}\boldsymbol{\mu}_n\\ -2\boldsymbol{\mu}_n\\ \boldsymbol{\phi}_1(\boldsymbol{\mu}_n)\\ \boldsymbol{\phi}_2(\boldsymbol{\Sigma}_n^{-\frac{1}{2}})\\ \boldsymbol{\Sigma}_n^{-1}\boldsymbol{\mu}_n\\ 1\end{bmatrix}}_{\mathbf{c}(\boldsymbol{\mu}_n)}\end{aligned}\tag{65}$$

For covariance prediction, consider the following formulation for the cost function:

$$\begin{aligned}&-\log|\boldsymbol{\Sigma}|+\mathtt{Tr}(\boldsymbol{\Sigma}_n^{-1}\boldsymbol{\Sigma})\\&=\underbrace{\begin{bmatrix}\log|\boldsymbol{\Sigma}|\\ \mathtt{vec}(\boldsymbol{\Sigma})\\ 1\end{bmatrix}}_{\mathbf{c}(\boldsymbol{\Sigma})}{}^\top\underbrace{\left[\begin{array}{c:c:c}\mathbf{0}&\mathtt{blkadiag}(&\\ &-1,\mathbf{I}_{d^2})&\\ \hdashline\mathbf{0}&&\mathbf{0}\end{array}\right]}_{V}\underbrace{\begin{bmatrix}\log|\boldsymbol{\Sigma}_n|\\ \mathtt{vec}(\boldsymbol{\Sigma}_n)\\ 1\end{bmatrix}}_{\mathbf{c}(\boldsymbol{\Sigma}_n)}\end{aligned}\tag{66}$$

B Definitions for Riemannian manifold

Here we provide basic notions and corresponding notations on Riemannian statistics that are used throughout this paper. We refer interested readers to Absil et al. (2009) for more details on manifolds. Informally, a d-dimensional Riemannian manifold $(\mathcal{M},g)$ is a topological space that locally behaves like the Euclidean space $\mathbb{R}^d$. Every point $\mathbf{p}\in\mathcal{M}$ has a tangent space $\mathcal{T}_\mathbf{p}\mathcal{M}$ where an inner product g is defined. When $\mathcal{M}$ is a *submanifold* of $\mathbb{R}^{d+1}$, the inner product can inherit from the standard Euclidean inner product in a natural way. The minimum distance between two points $\mathbf{p}_1$ and $\mathbf{p}_2$ on a Riemannian manifold is called *geodesic distance* $\mathtt{dist}(\mathbf{p}_1,\mathbf{p}_2)$, which generalizes the concept of straight lines in Euclidean spaces. The *exponential map* $\mathtt{Exp}_\mathbf{p}:\mathcal{T}_\mathbf{p}\mathcal{M}\to\mathcal{M}$ maps a point in the tangent space to the manifold with the distance and direction preserved. A *retraction* map $R_\mathbf{p}:\mathcal{T}_\mathbf{p}\mathcal{M}\to\mathcal{M}$ is a first order approximation of the exponential map. The inverse mapping of the exponential map is called the *logarithm map* $\mathtt{Log}_\mathbf{p}:\mathcal{M}\to\mathcal{T}_\mathbf{p}\mathcal{M}$. The logarithmic map is defined except for the *cut locus* of the base, which is the set of points that are connected by more than one geodesic curve with the base. *Parallel transport* $\Gamma_{\mathbf{p}_1\to\mathbf{p}_2}(\mathfrak{u}):\mathcal{T}_{\mathbf{p}_1}\mathcal{M}\to\mathcal{T}_{\mathbf{p}_2}\mathcal{M}$ moves vectors between tangent spaces such that the angles between the vectors and the geodesic curve connecting the bases are conserved. This operation is necessary to transport information available in one tangent space to another. The *Cartesian product* of two Riemannian manifolds $\mathcal{M}_1\times\mathcal{M}_2$ is still a Riemannian manifold and the corresponding manifold operations mentioned above can be obtained by concatenating the individual operations. A pictorial illustration is shown in Figure 12.

References

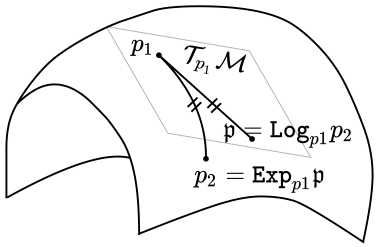
Abbeel P, Coates A and Ng AY (2010) Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research* 29(13): 1608–1639.

Abbeel P and Ng AY (2004) Apprenticeship learning via inverse reinforcement learning. In: *Proceedings of the twenty-first international conference on Machine learning*. p. 1.

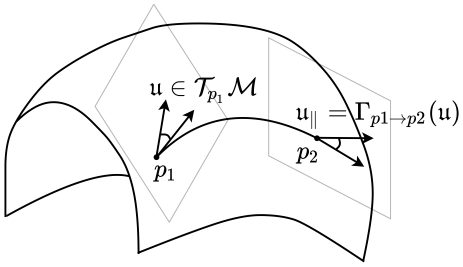
Absil PA, Mahony R and Sepulchre R (2009) *Optimization algorithms on matrix manifolds*. Princeton University Press.

Ahmadzadeh SR and Chernova S (2018) Trajectory-based skill learning using generalized cylinders. *Frontiers in Robotics and AI* 5.

Ajoudani A, Fang C, Tsagarakis N and Bicchi A (2018) Reduced-complexity representation of the human arm active endpoint stiffness for supervisory control of remote manipulation. *The International Journal of Robotics Research* 37(1): 155–167.



(a) Manifold exponential and logarithmic mappings.



(b) Parallel transport of vector.

Figure 12. Illustration of basic Riemannian notions.

Álvarez MA, Rosasco L and Lawrence ND (2012) Kernels for vector-valued functions: A review. *Foundations and Trends® in Machine Learning* 4(3): 195–266.

Amanhoud W, Khoramshahi M and Billard A (2019) A dynamical system approach to motion and force generation in contact tasks. Robotics: Science and Systems (RSS).

Arduengo M, Colomé A, Borràs J, Sentis L and Torras C (2021) Task-adaptive robot learning from demonstration with Gaussian process models under replication. *IEEE Robotics and Automation Letters* 6(2): 966–973. DOI:10.1109/LRA.2021.3056367.

Bahl S, Mukadam M, Gupta A and Pathak D (2020) Neural dynamic policies for end-to-end sensorimotor learning. *Advances in Neural Information Processing Systems* 33: 5058–5069.

Beik-Mohammadi H, Hauberg S, Arvanitidis G, Neumann G and Rozo L (2021) Learning Riemannian manifolds for geodesic motion skills. In: *Robotics: Science and Systems*.

Billard A, Calinon S, Dillmann R and Schaal S (2008) Robot programming by demonstration. *Springer handbook of robotics* : 1371–1394.

Bonalli R, Bylard A, Cauligi A, Lew T and Pavone M (2019) Trajectory optimization on manifolds: A theoretically-guaranteed embedded sequential convex programming approach. In: *Robotics: Science and Systems*.

Calinon S (2020) Gaussians on riemannian manifolds: Applications for robot learning and adaptive control. *IEEE Robotics & Automation Magazine* 27(2): 33–45.

Cheng CA, Yan X, Wagener N and Boots B (2018) Fast policy learning through imitation and reinforcement. In: *Uncertainty in artificial intelligence*.

Ciliberto C, Rosasco L and Rudi A (2016) A consistent regularization approach for structured prediction. In: *Advances in neural information processing systems*. pp. 4412–4420.

Ciliberto C, Rosasco L and Rudi A (2020) A general framework for consistent structured prediction with implicit loss embeddings. *J. Mach. Learn. Res.* 21(98): 1–67.

Darvish K, Penco L, Ramos J, Cisneros R, Pratt J, Yoshida E, Ivaldi S and Pucci D (2023) Teleoperation of humanoid robots: A survey. *IEEE Transactions on Robotics* : 1–22DOI:10.1109/TRO.2023.3236952.

Duan A, Camoriano R, Ferigo D, Calandriello D, Rosasco L and Pucci D (2018) Constrained DMPs for feasible skill learning on humanoid robots. In: *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*. IEEE, pp. 1–6.

Duan A, Camoriano R, Ferigo D, Huang Y, Calandriello D, Rosasco L and Pucci D (2019) Learning to sequence multiple tasks with competing constraints. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 2672–2678.

Duan A, Camoriano R, Ferigo D, Huang Y, Calandriello D, Rosasco L and Pucci D (2020) Learning to avoid obstacles with minimal intervention control. *Frontiers in Robotics and AI* 7: 60.

Duan A, Victorova M, Zhao J, Sun Y, Zheng Y and Navarro-Alarcon D (2022) Ultrasound-guided assistive robots for scoliosis assessment with optimization-based control and variable impedance. *IEEE Robotics and Automation Letters* 7(3): 8106–8113.

Figueroa N and Billard A (2018) A physically-consistent bayesian non-parametric mixture model for dynamical system learning. In: *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings*, volume 87.

Florence P, Lynch C, Zeng A, Ramirez OA, Wahid A, Downs L, Wong A, Lee J, Mordatch I and Tompson J (2022) Implicit behavioral cloning. In: *Conference on Robot Learning*. PMLR, pp. 158–168.

Ghasemipour SKS, Zemel R and Gu S (2020) A divergence minimization perspective on imitation learning methods. In: *Conference on Robot Learning*. PMLR, pp. 1259–1277.

Ho J and Ermon S (2016) Generative adversarial imitation learning. In: *Advances in neural information processing systems*. pp. 4565–4573.

Huang Y, Rozo L, Silvério J and Caldwell DG (2019) Kernelized movement primitives. *The International Journal of Robotics Research* 38(7): 833–852.

Ijspeert AJ, Nakanishi J, Hoffmann H, Pastor P and Schaal S (2013) Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation* 25(2): 328–373.

Ke L, Choudhury S, Barnes M, Sun W, Lee G and Srinivasa S (2021) Imitation learning as f-divergence minimization. In: *Algorithmic Foundations of Robotics XIV: Proceedings of the Fourteenth Workshop on the Algorithmic Foundations of Robotics 14*. Springer International Publishing, pp. 313–329.

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

Khoramshahi M, Henriks G, Naef A, Salehian SSM, Kim J and Billard A (2020) Arm-hand motion-force coordination for physical interactions with non-flat surfaces using dynamical systems: Toward compliant robotic massage. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE.

Kingston Z, Moll M and Kavraki LE (2019) Exploring implicit spaces for constrained sampling-based planning. *The International Journal of Robotics Research* 38(10-11): 1151–1178.

Kober J and Peters J (2014) Policy search for motor primitives in robotics. In: *Learning Motor Skills*. Springer, pp. 83–117.

Kronander K and Billard A (2015) Passive interaction control with dynamical systems. *IEEE Robotics and Automation Letters* 1(1): 106–113.

Kulvicius T, Ning K, Tamosiunaite M and Worgötter F (2011) Joining movement sequences: Modified dynamic movement primitives for robotics applications exemplified on handwriting. *IEEE Transactions on Robotics* 28(1): 145–157.

Mroueh Y, Poggio T, Rosasco L and Slotine JJ (2012) Multiclass learning with simplex coding. In: *Advances in Neural Information Processing Systems*. pp. 2789–2797.

Osa T, Pajarinen J, Neumann G, Bagnell JA, Abbeel P and Peters J (2018) An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics* 7(1-2): 1–179.

Paraschos A, Daniel C, Peters JR and Neumann G (2013) Probabilistic movement primitives. In: *Advances in neural information processing systems*. pp. 2616–2624.

Pardo L (2018) *Statistical inference based on divergence measures*. Chapman and Hall/CRC.

Peng XB, Abbeel P, Levine S and van de Panne M (2018) Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)* 37(4): 143.

Peters J, Mulling K and Altun Y (2010) Relative entropy policy search. In: *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Pomerleau DA (1989) Alvinn: An autonomous land vehicle in a neural network. In: *Advances in neural information processing systems*. pp. 305–313.

Ratliff ND, Silver D and Bagnell JA (2009) Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots* 27(1): 25–53.

Ravichandar H, Polydoros AS, Chernova S and Billard A (2020) Recent advances in robot learning from demonstration. *Annual Review of Control, Robotics, and Autonomous Systems* 3.

Reiner B, Ertel W, Posenauer H and Schneider M (2014) LAT: A simple learning from demonstration method. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 4436–4441.

Ross S, Gordon G and Bagnell D (2011) A reduction of imitation learning and structured prediction to no-regret online learning. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. pp. 627–635.

Ruan S, Poblete KL, Wu H, Ma Q and Chirikjian GS (2023) Efficient path planning in narrow passages for robots with ellipsoidal components. *IEEE Transactions on Robotics* 39(1): 110–127. DOI:10.1109/TRO.2022.3187818.

Rudi A, Camoriano R and Rosasco L (2015) Less is more: Nyström computational regularization. *Advances in Neural Information Processing Systems* 28.

Rudi A, Ciliberto C, Marconi G and Rosasco L (2018) Manifold structured prediction. In: *Advances in Neural Information Processing Systems*. pp. 5610–5621.

Schaal S (1999) Is imitation learning the route to humanoid robots? *Trends in cognitive sciences* 3(6): 233–242.

Schneider M and Ertel W (2010) Robot learning by demonstration with local Gaussian process regression. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 255–260.

Schulman J, Levine S, Abbeel P, Jordan M and Moritz P (2015) Trust region policy optimization. In: *International conference on machine learning*. pp. 1889–1897.

Simo-Serra E, Torras C and Moreno-Noguer F (2017) 3D human pose tracking priors using geodesic mixture models. *International Journal of Computer Vision* 122(2): 388–408.

Stulp F and Sigaud O (2015) Many regression algorithms, one unified model: A review. *Neural Networks* 69: 60–79.

Swamy G, Choudhury S, Bagnell JA and Wu S (2021) Of moments and matching: A game-theoretic framework for closing the imitation gap. In: *International Conference on Machine Learning*. PMLR, pp. 10022–10032.

Traversaro S, Brossette S, Escande A and Nori F (2016) Identification of fully physical consistent inertial parameters using optimization on manifolds. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 5446–5451.

Yang C, Zeng C, Fang C, He W and Li Z (2018) A DMPs-based framework for robot learning and generalization of humanlike variable impedance skills. *IEEE/ASME Transactions on Mechatronics* 23(3): 1193–1203.

Zahra O, Tolu S, Zhou P, Duan A and Navarro-Alarcon D (2022) A bio-inspired mechanism for learning robot motion from mirrored human demonstrations. *Frontiers in Neurorobotics* 16.

Zeestraten MJ, Havoutis I, Calinon S and Caldwell DG (2017a) Learning task-space synergies using Riemannian geometry. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 73–78.

Zeestraten MJ, Havoutis I, Silvério J, Calinon S and Caldwell DG (2017b) An approach for imitation learning on Riemannian manifolds. *IEEE Robotics and Automation Letters* 2(3): 1240–1247.

Zhou Y and Asfour T (2017) Task-oriented generalization of dynamic movement primitive. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 3202–3209.

Ziebart BD, Maas A, Bagnell JA and Dey AK (2008) Maximum entropy inverse reinforcement learning. In: *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08. AAAI Press. ISBN 9781577353683, p. 1433–1438.