Received 7 October 2022, accepted 24 October 2022, date of publication 28 October 2022, date of current version 4 November 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3217800

RESEARCH ARTICLE

Learning Deep Robotic Skills on Riemannian Manifolds

WEITAO WANG[1], **MATTEO SAVERIANO**[2], **(Member, IEEE), AND FARES J. ABU-DAKKA**[1], **(Member, IEEE)**
[1]Intelligent Robotics Group, Department of Electrical Engineering and Automation (EEA), Aalto University, 02150 Espoo, Finland
[2]Department of Industrial Engineering (DII), University of Trento, 38123 Trento, Italy

Corresponding author: Fares J. Abu-Dakka (fares.abu-dakka@aalto.fi)

This work was supported in part by CHIST-ERA Project IPALM (Academy of Finland Decision) under Grant 326304.

ABSTRACT In this paper, we propose RiemannianFlow, a deep generative model that allows robots to learn complex and stable skills evolving on Riemannian manifolds. Examples of Riemannian data in robotics include stiffness (symmetric and positive definite matrix (SPD)) and orientation (unit quaternion (UQ)) trajectories. For Riemannian data, unlike Euclidean ones, different dimensions are interconnected by geometric constraints which have to be properly considered during the learning process. Using distance preserving mappings, our approach transfers the data between their original manifold and the tangent space, realizing the removing and re-fulfilling of the geometric constraints. This allows to extend existing frameworks to learn stable skills from Riemannian data while guaranteeing the stability of the learning results. The ability of RiemannianFlow to learn various data patterns and the stability of the learned models are experimentally shown on a dataset of manifold motions. Further, we analyze from different perspectives the robustness of the model with different hyperparameter combinations. It turns out that the model's stability is not affected by different hyperparameters, a proper combination of the hyperparameters leads to a significant improvement (up to 27.6%) of the model accuracy. Last, we show the effectiveness of RiemannianFlow in a real peg-in-hole (PiH) task where we need to generate stable and consistent position and orientation trajectories for the robot starting from different initial poses.

INDEX TERMS Compliance and impedance control, deep learning methods, learning from demonstration, motion control of manipulators, Riemannian manifold.

I. INTRODUCTION

Robots operating in everyday environments nowadays are required not only to follow some rigid movements, but also to accomplish complex physical interactions with the environment, including tools, humans, and/or other robots. During these activities, the reference position, stiffness, and orientation are changing along with the procedure, which makes it extremely time-consuming to implement with traditional hard-programming methods. In such a situation, Learning from Demonstration (LfD) [1] is considered as a more efficient way for robots to acquire new abilities, as even people with no robotics knowledge are able to give demonstrations teaching robots their own specialties (see Fig.1). The learning-based solution proposed in this work can be directly exploited by robotics experts and non-experts to accomplish desired tasks without complex programming, which will accelerate the transition from ideas to products in manufacturing and save much cost at the same time.

The Trajectory-tracking task is an active research topic in robotics, which is indeed a fundamental component in a LfD system. In these path-following tasks, the main focus is to generate a robust and precise route *w.r.t.*[1] the position and orientation [2], [3]. However, to achieve more complicated motions, different kinds of control information like force

The associate editor coordinating the review of this manuscript and approving it for publication was Zhong Wu.

[1]*w.r.t.*stands for *with respect to*.

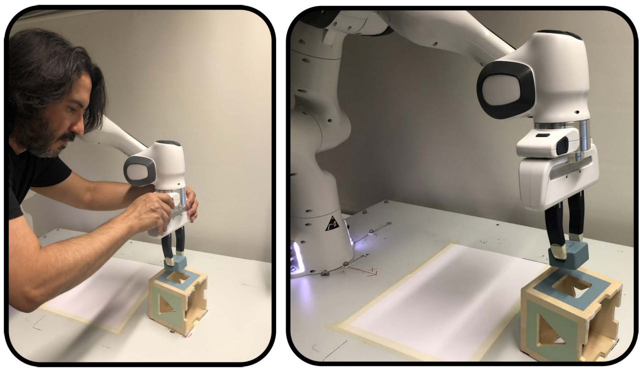


FIGURE 1. *Left*: A human operator teaches a robot how to perform a PiH task. *Right*: a snapshot showing the PiH task executed by a Franka Emika Panda robot.

and stiffness [4] are regarded as objects to be learned from demonstrations. Unlike simple data, e.g., data of position or joint angles where Degrees-of-Freedoms (DoFs) are independent, some data are represented by special formats. For instance, when we handle stiffness, inertia, or sensory data organized as covariance features, Symmetric Positive Definite (SPD) matrices are encountered, while Unit Quaternions (UQs) are used to represent orientations. For such data, DoFs are related by particular constraints arising from the structure of the underlying space, which makes the learning problem more difficult as the learner has also to fulfill those constraints.

In this paper, we propose a geometry-aware approach to correctly handle complex data structures and their underlying constraints. In particular, we focus on data evolving on Riemannian Manifolds (RMs). As handling geometric constraints directly on the RM is complex, we exploit a distance preserving mapping to move the data from a manifold to a local tangent space. As the tangent space is a finite dimensional linear space, it is isomorphic to the Euclidean space and tools from linear algebra can be applied freely on the transformed data. On the tangent space, the transformed data are considered as generated from a stochastic nonlinear dynamical system whose dynamics is learned using the ImitiationFlow approach [5]. The learned model is used at run-time to retrieve the data on the tangent space which are then projected back to the original RM using a reverse mapping. The inverse mapping automatically imposes constraints of the geometric structure of the manifold on the tangent space motion. A similar approach is used in [6] to learn stable UQ motions. The main difference is that [6] works on Lie groups like UQ, while our approach is more general as it works also on manifolds like SPD matrices that are not a Lie group. In our experiments, the proposed RiemannianFlow is proved to be able to accurately learn stable dynamical systems evolving on different, possibly high dimensional RMs. To summarize, our contribution is three-fold:

- We propose a geometry-aware approach to learn stable robotic skills on RMs.
- We show that our approach allows to straightforwardly extend existing approaches, including modern deep learning techniques, to learn from manifold data.
- We compare our approach with two baselines and two state-of-the-art approaches on a public benchmark.

The rest of the paper is organized as follows. Section II presents the related work and highlights similarities and differences with our method. In Sec. III, basic knowledge of RM and the LfD model used in this work are introduced. Section IV presents the proposed approach. We compare RiemannianFlow with two baseline and two state-of-the-art approaches in Sec. V and show an experiment in a typical industrial task. Section VII states the conclusions and proposes further research directions.

II. RELATED WORK

For robot-environment interaction, robots need to have a sophisticated adaptation of their end-effector pose and stiffness. The importance of controlling pose and stiffness for the successful accomplishment of robotic tasks makes them fundamental research topics in the field of robot manipulation.

For the stiffness, an intuitive method was provided by [7] to generate the stiffness directly from the limb postures and surface Electromyography (EMG) signals of the human demonstrator. Authors in [8] used human demonstration along with EMG to learn stiffness from human muscle activity measurements. The learning framework proposed in [4] used the measured trajectories and interaction forces to derive a full stiffness matrices profile encoded in a probabilistic model to be executed later with unseen situations. Kronander and Billard [9] exploited the variability in the demonstrations, injected by shacking the robot during the motion, to learn a variable stiffness profile where high variance corresponds to low stiffness (less accurate tracking). Kernelized Movement Primitives (KMPs) [10] were exploited in [11] to generalize variable stiffness profiles. Jaquier et al. [12] proposed a Gaussian Mixture Model (GMM)-based framework to learn SPD profiles with a particular focus on robot manipulability.

For the orientation, early work [13] did not consider geometric constraints—unit norm for UQs or orthogonality for rotation matrices—when they were learning the orientation data. Instead, they modified the generated trajectory at run-time to fulfill the constraints, causing deviations from the demonstrated motion. To remedy this issue, the work in [14], [15] extended the classical dynamic movement primitives (DMP) formulation [16] to properly handle orientation data. The stability of the obtained approach is shown in [17] for UQs. Abu-Dakka et al. [18] extended periodic DMPs to encode periodic orientation patterns. GMMs and Task-Parameterized Gaussian Mixture Models (TP-GMMs) were extended to describe the distribution of UQs in [19] and [20], respectively. Rozo and Dave [21] proposed a Riemannian extension of the probabilistic movement primitives framework. KMPs were also extended to represent orientation trajectories by exploiting the mappings between UQ manifold and its tangent space [22].

RiemannianFlow exploits the idea of projecting data in the tangent space to remove geometric constraints and then re-fulfilling them by projecting the generated data back to the manifold. Working in the tangent space allows to use standard tools from Euclidean geometry to encode the demonstrations and fit a stable dynamics in the tangent space to ensure convergence to a given target on the manifold. Moreover, it makes the formulation relatively general and easy to adapt to different manifolds. Indeed, the major change is to use the proper projection operations which are manifold dependent. It is worth mentioning that working in the tangent space is a local approach that is exact only if the data belong to the same chart on the manifold. This problem affects UQs, but not SPD matrices [23]. As shown in [24], if quaternion data span multiple charts then using a single tangent space introduces significant approximation errors. Instead of using multiple tangent spaces as in [24], which requires clustering of the data, we use a simple pre-processing step to ensure that quaternion data belong to the same hemisphere (see Sec. IV-A).

III. BACKGROUND

This section briefly describes the geometric structure of SPD, UQ manifolds and the key aspects of ImitationFlow [5].

A. DISTANCE PRESERVING TRANSFORMATIONS ON rms

In this paper, we focus mainly on two RMs, namely the $d \times d$ SPD matrices $\mathcal{S}_d^+$ and the unit-sphere $\mathcal{S}^3$ (UQs). As already mentioned, the RiemannianFlow works by moving manifold data into the local tangent space $\mathcal{TM}$ and back to the manifold $\mathcal{M}$. We are also interested in preserving the convergence of the learned motion to a given point on the manifold, namely the goal $\boldsymbol{g} \in \mathcal{M}$. Therefore, we consider the tangent space at the goal $\mathcal{T}_{\boldsymbol{g}}\mathcal{M}$ and project our data on it. For this, we need to define the logarithmic mapping function $\text{Log}_{\boldsymbol{g}}(\boldsymbol{p}) : \mathcal{M} \rightarrow \mathcal{T}_{\boldsymbol{g}}\mathcal{M}$ which moves a point $\boldsymbol{p}$ from the manifold to $\mathfrak{p}$ on the tangent space of $\boldsymbol{g}$, along the projection of the geodesic (the shortest curve on the RM) between $\boldsymbol{p}$ and $\boldsymbol{g}$. In this way, data on the RM can be transferred to the tangent space to be manipulated freely. After the learning procedure, the exponential mapping function $\text{Exp}_{\boldsymbol{g}}(\mathfrak{p}) : \mathcal{T}_{\boldsymbol{g}}\mathcal{M} \rightarrow \mathcal{M}$, which is the inverse of the logarithmic mapping, can transfer the data back to the RM.

For the manifold $\mathcal{S}_d^+$, $\boldsymbol{p}$ and $\boldsymbol{g}$ are represented by SPD matrices and $\mathfrak{p}$ is described by a symmetric matrix, these 2 mappings can be computed as [23]:

$$\text{Log}_{\boldsymbol{g}}(\boldsymbol{p}) = \boldsymbol{g}^{\frac{1}{2}} \text{logm}(\boldsymbol{g}^{-\frac{1}{2}}\boldsymbol{p}\boldsymbol{g}^{-\frac{1}{2}})\boldsymbol{g}^{\frac{1}{2}}, \tag{1}$$

$$\text{Exp}_{\boldsymbol{g}}(\mathfrak{p}) = \boldsymbol{g}^{\frac{1}{2}} \text{expm}(\boldsymbol{g}^{-\frac{1}{2}}\mathfrak{p}\boldsymbol{g}^{-\frac{1}{2}})\boldsymbol{g}^{\frac{1}{2}}, \tag{2}$$

where logm(·) and expm(·) are the matrix logarithm and exponential respectively. Last, we use Mandel's notation to transform the symmetric matrix $\mathfrak{p}$ to a vector vec($\mathfrak{p}$) and vice versa. Therefore, the whole process realizes the conversion between SPD matrices and vectors.

For the manifold $\mathcal{S}^3$, $\boldsymbol{p}$ is represented by a UQ, where $\boldsymbol{p} = \nu + \boldsymbol{u}$, and $\mathfrak{p}$ is described by a 3-dimensional vector. Therefore, the logarithmic and exponential mappings [14] are:

$$\text{Log}_{\boldsymbol{g}}(\boldsymbol{p}) = \text{Log}(\boldsymbol{p} * \bar{\boldsymbol{g}}) = \begin{cases} \arccos(\nu)\dfrac{\boldsymbol{u}}{||\boldsymbol{u}||}, & \boldsymbol{u} \neq \mathbf{0} \\ [0 \;\; 0 \;\; 0]^{\top}, & \text{otherwise.} \end{cases} \tag{3}$$

$$\text{Exp}_{\boldsymbol{g}}(\mathfrak{p}) = \begin{cases} \left[\cos(||\mathfrak{p}||) + \sin(||\mathfrak{p}||)\dfrac{\mathfrak{p}}{||\mathfrak{p}||}\right] * \boldsymbol{g}, & \mathfrak{p} \neq \mathbf{0} \\ \left[1 + [0 \;\; 0 \;\; 0]^{\top}\right] * \boldsymbol{g}, & \text{otherwise.} \end{cases} \tag{4}$$

where $*$ denotes UQs multiplication as in [14].

B. FLOW-BASED DYNAMIC MODEL

ImitationFlow [5] assumes that observations are sampled from an unknown, arbitrary complex distribution $\mathfrak{p} \sim \psi(\mathfrak{p})$ that is related to a latent, known distribution $\mathfrak{q} \sim \pi(\mathfrak{q})$ through a diffeomorphic—continuous, bijective, and with continuous derivative—map $\mathrm{b}(\cdot) : \mathbb{R}^d \rightarrow \mathbb{R}^d$. Specifically, having the latent distribution $\mathfrak{q} \sim \pi(\mathfrak{q})$, a different distribution $\mathfrak{p} \sim \psi(\mathfrak{p})$ is acquired by $\mathfrak{p} = \mathrm{b}(\mathfrak{q})$.

In a LfD setting, observations of the complex distribution $\psi(\mathfrak{p})$ are given in the form of expert demonstrations and can be used to learn the diffeomorphism b(·), e.g., using a Normalizing Flow [25]. To enforce the stability of the learned motion, ImitationFlow assumes that the transition model in the latent space follows a known stable dynamics. The simplest stable stochastic dynamics is given by

$$\dot{\mathfrak{q}}(t) = V_{\phi}\mathfrak{q}(t) + F_{\phi}\boldsymbol{\beta}(t), \tag{5}$$

where $\mathfrak{q} \in \mathbb{R}^d$ represents the state and $\boldsymbol{\beta} : \mathbb{R} \rightarrow \mathbb{R}^d$ (called Brownian motion or Wiener process) represents noise and variability in the demonstrations. Matrices V_{ϕ} and F_{ϕ} are parameterized by the set of parameters ϕ. To ensure stability of the base dynamics, ϕ are constrained such that the eigenvalues of V_{ϕ} have negative real part. The output of (5) is

$$\mathfrak{p} = \mathrm{b}_{\theta}(\mathfrak{q}), \tag{6}$$

where b_{θ} is the diffeomorphic mapping parameterized by θ. The free parameters of the model are θ and ϕ which can be learned from the given demonstrations. As shown in [5], the observation space dynamics obtained by differentiating (6) is stable if the latent space dynamics is stable.

The presented approach assumes that both the latent and the observation space are isomorphic to an Euclidean space. In the following section, we are discussing how to extend flow-based models to RMs.

IV. PROPOSED APPROACH

The problem of LfDs on stiffness or orientation profiles is to properly consider the geometric constraints imposed on these complex data structures. To tackle it, a geometry-aware approach is proposed considering that geometric constraints

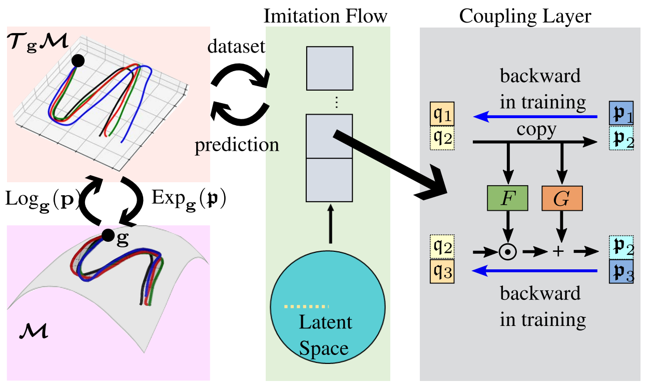


FIGURE 2. A pipeline indicates the whole process of the methods. Note: For each coupling layer, the second dimension is either in the upper group or in the lower group, and between each coupling layer there is a swap among dimensions to enlarge the capacity. For the 2 transformation functions: $F, G : \mathbb{R}^d \to \mathbb{R}^d$, they are implemented by multi-layer perceptron.

"disappear" when data are transferred from the RM to the local tangent space. Therefore, once the data are transferred to the tangent space, standard tools can be used to learn the demonstrated patters. We choose to learn a stable trajectory in the tangent space by applying a diffeomorphic transformation on a linear and stable dynamics. To learn the diffeomorphism, we use the deep generative model proposed in [5]. Last, we apply the the reverse mapping to project the generated data back to the RM which imposes geometric constraints on the generated data. An in-depth description of our RiemannianFlow approach is given in the rest of this section.

A. ACQUIRE TRAINING DATA FROM DEMONSTRATION

We want to learn the diffeomorphic mapping b_θ, parameterized by θ, that maps the trajectory of a simple dynamics into a more complex, desired one. As commonly done in LfDs, we assume that a set of $N \geq 1$ demonstrations of the same skill is given, i.e., $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N\}$, where each demonstration $\mathcal{D}_n$ contains M points on a RM, i.e., $\mathcal{D}_n = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_M\}$, and each point $\boldsymbol{p}_m \in \mathcal{S}_d^+$ or $\boldsymbol{p}_m \in \mathcal{S}^3$, $m = 1, \ldots, M$. For simplicity, the sampling time δt is assumed to be the same for all the demonstrations. Since we are interested in learning converging motions, we also assume that all the demonstrations converge to the same goal, i.e., $\boldsymbol{p}_M = \boldsymbol{g}$, $\forall \mathcal{D}_n$.

The first pre-processing step, needed only if $\boldsymbol{p}_m \in \mathcal{S}^3$, is to check whether the dot product $\boldsymbol{p}_m \cdot \boldsymbol{p}_{m+1} > 0$. In case the dot product is negative, we substitute $\boldsymbol{p}_{m+1}$ with $-\boldsymbol{p}_{m+1}$ to prevent discontinuities in the quaternion trajectory. As a result of this step, we ensure that the entire quaternion trajectory is contained in a single chart (hemisphere) where logarithmic and exponential mappings are bijective. It is worth mentioning that a similar step is not needed for $\mathcal{S}_d^+$ since logarithmic and exponential maps are bijective everywhere inside the SPD cone. After this check, we project all the points in the tangent space by means of the logarithmic mapping $\mathrm{Log}_{\boldsymbol{g}}(\boldsymbol{p}_m)$, which is defined in (1) for SPD and in (3) for UQ. Only for SPD manifold, we vectorize the data on the tangent space using Mandel's notation. The result is the new set of demonstrations with the same structure of the original ones but on the tangent space. It is worth noticing that, having placed the tangent space in the last point of the demonstrations, it holds that $\mathfrak{p}_M = \mathrm{Log}_{\boldsymbol{g}}(\boldsymbol{p}_M) = \mathrm{Log}_{\boldsymbol{g}}(\boldsymbol{g}) = \mathbf{0}$.

B. DEEP STABLE DYNAMICS ON RMs

Using the approach presented in the previous section we transform the given demonstrations into a new set of demonstrations $\mathcal{T}$ containing points in the tangent space. Given the tangent space is a finite dimensional linear space, where points can be vectorized without loss of information, it is possible to exploit existing approaches to learn a stable dynamics in the tangent space. Since our targets manifolds contain complex objects like stiffness or orientation of robots, not only the precision of the learning results matters, but also the robustness. Therefore, a generative model based on probability is chosen for its advantage of generality. In particular, we adopt a flow-based model (Sec. III-B) to fit a stable stochastic dynamic in the tangent space. The learned model can be directly used to generate stable tangent space trajectories, as shown in Fig. 4.

To generate a Riemannian trajectory, we proceed as follows. Given an initial point $\boldsymbol{p}_1 \in \mathcal{S}_d^+$ (or $\boldsymbol{p}_1 \in \mathcal{S}^3$) and a desired goal point $\boldsymbol{g} \in \mathcal{S}_d^+$ (or $\boldsymbol{g} \in \mathcal{S}^3$), we use the logarithmic mapping to project $\boldsymbol{p}_1$ onto the tangent space centered at $\boldsymbol{g}$, obtaining $\mathfrak{p}_1 = \mathrm{Log}_{\boldsymbol{g}}(\boldsymbol{p}_1)$. Then, we pass $\mathfrak{p}_1$ through the learned tangent space dynamic and numerically integrate the generated velocity to obtain $\mathfrak{p}_2$. Last, we project back the point onto the manifold by means of the exponential mapping, obtaining $\boldsymbol{p}_2 = \mathrm{Exp}_{\boldsymbol{g}}(\mathfrak{p}_2)$. The last step is fundamental as it re-imposes constraints on the generated point preserving the geometric structure of the manifold. We repeat until the goal is reached. To check for convergence, we recall that, having placed the tangent space in the goal, the tangent space trajectory should converge to zero. Therefore, we simply stop the procedure when the generated point $\|\mathfrak{p}_m\| < \xi \approx 0$, where $\|\cdot\|$ is the Euclidean norm and ξ is a user defined threshold. The proposed approach to generate stable motions on RMs is summarized in Algorithm 1 and can be visualized in Fig. 2.

V. EXPERIMENTAL EVALUATION

In this section, we test RiemannianFlow both in simulations and a real experiment. For the simulations, we augment a popular dataset [26] with UQ and SPD data. For the experiment, we use a 7 degrees-of-freedom Franka Emika Panda manipulator to perform a PiH task.

A. DATASET CREATION

The dataset we used is based on LASA dataset [26] which consists of 30 shapes, each with 7 trajectories of 1000 points in 2- dimensional. We recombined the dimensions of different trajectories of the same shape and got 4 3-dimensional

Algorithm 1 RiemannianFlow
```
1: Compute training data
   – Collect demonstrations $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N\}$ of Riemannian data $\mathcal{D}_n = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_M\}$
   – Define the goal $\boldsymbol{g}$, e.g., the last point of each $\mathcal{D}_n$.
   – Check if $\mathcal{S}^3$ data are in the same chart: $\forall \boldsymbol{p}_m \in \mathcal{D}_n \rightarrow \boldsymbol{p}_m \cdot \boldsymbol{p}_{m+1} > 0$. If not, substitute $\boldsymbol{p}_{m+1}$ with $-\boldsymbol{p}_{m+1}$
   – Project all the points to the tangent space using the logarithmic map:
     $\forall \boldsymbol{p}_m \in \mathcal{D}_n \rightarrow \mathfrak{p}_m = \mathrm{Log}_{\boldsymbol{g}}(\boldsymbol{p}_m)$.
     Vectorize $\mathcal{S}_d^+$ data (e.g., using Mandel's notation)
   – Save tangent space data into the set of demonstrations $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_N\}$
2: Learn a stable motion in tangent space, e.g., with ImitationFlow [5], using $\mathcal{T}$ as training data.
3: Generate Riemannian trajectories
      $\mathfrak{p}_m \leftarrow \mathrm{Log}_{\boldsymbol{g}}(\boldsymbol{p}_1)$
      while $\|\mathfrak{p}_m\| \geq \xi$ do
          $\dot{\mathfrak{p}}_m \leftarrow \mathrm{ImitationFlow}(\mathfrak{p}_m)$
          $\boldsymbol{p}_m \leftarrow \mathrm{Exp}_{\boldsymbol{g}}(\dot{\mathfrak{p}}_m \delta t)$
      end while
```

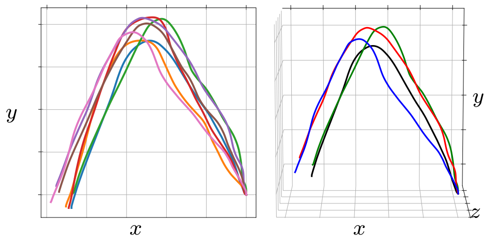


FIGURE 3. ***Left***: **"A" shape of original LASA data.** ***Right***: **"A" shape of created data.**

trajectories for each shape. More in detail, we first stacked the 7 demonstrations of each shape in a matrix $\boldsymbol{D}_i$ for $i = 1, \ldots, 30$ with 14 rows and 1000 columns. Given $\boldsymbol{D}_i$, we extracted the 4 demonstrations by selecting the rows [0, 1, 2], [4, 5, 6], [8, 9, 10], and [12, 3, 0]. In this way, the obtained 4 demonstrations have the third dimension sampled from the x-axis of the original data. This implies that the added dimension contains similar patterns for the same shape, as usually required in LfD.

One example of "A" shape data is shown in Fig. 3. We consider those 3D data as projections in the tangent space of UQ from which one can directly compute UQ profiles using (4). For SPD matrices, we assume that the 3D data represent the vectorization of symmetric 2×2 matrices obtained with Mandel's notation. We then invert Mandel's notation to compute 2×2 symmetric matrices (in tangent space) and retrieve SPD matrices using the exponential mapping in (2). The center of the tangent space was placed at $\boldsymbol{g}_{\mathrm{SPD}} = \mathrm{diag}([100, 100])$ for SPD matrices and at $\boldsymbol{g}_{\mathrm{q}} = 1 + [0, 0, 0]$ for UQs. Before training, we normalized each dimension of the tangent space data to zero mean and unit variance.

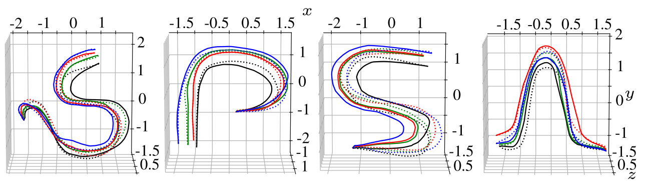


FIGURE 4. Visualization of data generated by our model (solid line) and given demonstrations (dot line) on the tangent space.

B. TRAINING PROCESS

Being each shape significantly different (see Fig. 8), we trained different models for each of the 30 shapes in the dataset. We randomly chose several shapes of our dataset to test the performance of the trained model. For each shape, the corresponding 4 demonstrations were used as the dataset for training.

In this stage, the number of coupling layers with random initialization was 10, playing the role of the emission function. Generally, after 40 epochs of training, the generated trajectories could reproduce accurately the given pattern. Further, all trajectories converged to the goal point in the end, which empirically proves the stability of the model. Examples of these data are shown in Fig. 4, where all the trajectories were generated by the model trained for only 40 epochs.

During the training process, Dynamic Time Warping (DTW) [27], indicating the similarity between the generated trajectories and given demonstrations, was considered as the evaluation factor and it was computed using the implementation provided by [28]. We monitored the value changes of DTW during 200 epochs of training for all 30 shapes, the value reached the lowest before 100 epochs in 25 cases out of 30. Therefore, for hyperparameter search, the maximum number of training epoch was limited to 100 to accelerate the process. Typical loss curves trained for 200 epochs are shown in Fig. 5.

C. HYPERPARAMETERS SEARCH

There are several hyperparameters in our model, including the number of layers, the activation function, the learning rate, and the optimizer. But before searching for the best combination, we first tested some different initialization methods to make sure the inside parameters were set correctly in the beginning. We tried popular initialization methods like He et al. [29] and found out that randomly uniform initialization made the training process to converge faster. Therefore, we used it in the following search process.

To make sure that different hyperparameter combinations affecting on the same model we set a certain seed for random initialization and chose the data of "N" shape to train the model. The search method used is provided by Optuna package [30] and it automatically prunes improper combinations based on the evaluation values during the search and generates the combination that tends to achieve a better performance, which can accelerate the whole process significantly.

TABLE I. Hyperparameters considered in the search, their initial range, and the best value found.

PARAMETER	OPTIONS	BEST VALUE
number of layers	8 to 12	11
activation function	ReLu, Tanh	ReLu
optimizer	Adam, Adamax, SGD, RMSprop	Adam
learning rate	1e-5 to 1e-1	0.00098

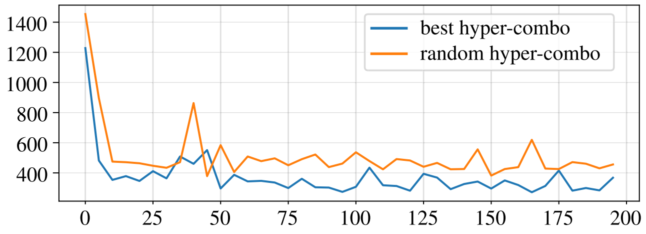


FIGURE 5. Loss curves indicating training process for "N" shape dataset. Loss is computed every 5 epochs. The orange curve indicates the training process under a random hyperparameter combination while the blue curve shows the loss trend with the best combination searched in Sec.V-C.

During the first rough search, the options for the hyperparameters are shown in Table I. We generated 100 combinations among which only 37 combinations were executed completely and the rest were pruned. From the search results, we noticed that for the activation function ReLu beats the other one; for the learning rate, the range shrinks to 1e-4 to 1e-2; and for the optimizer, Adam and Adamax [31] stand out. Also, we found Adamax was able to generate more stable training processes while Adam tended to reach lower minimum of errors.

Based on the above results, we carried out several rounds of fine search only utilizing ReLu as the activation function and just 2 options (Adam or Adamax) for the optimizer and the shrunk range for learning rate. However, since the random initialization shall also affect the training process, we gave different random seeds for different search processes. With this procedure and a seed of 20, we found the best hyperparameter configuration in Tab. I resulting in a DTW error of 274. Figure. 5 shows the learning curve with and without hyperparameters search. Moreover, the figure shows that the lower training error is achieved with the best parameters.

Regarding training and generation time, using the best hyperparameter setting in Tab. I, a batch size of 128 and 100 training epochs, and considering that a motion trajectory contains 4000 samples (3D vector), the average training time is 61 minutes for one shape utilizing a NVIDIA RTX 2060 GPU. Our approach takes only 20 s to generate an entire trajectory of 1000 samples.

For the best hyperparameters, we did further testing by transferring the generated data on the tangent space back to the manifold ($\mathcal{S}_2^+$ and $\mathcal{S}^3$) and comparing the distance between generated and given data. For SPD matrices, we use the Log Euclidean distance (LEd) [23]:

$$\mathrm{LEd} = \left\| \mathrm{logm}(\boldsymbol{s}) - \mathrm{logm}(\hat{\boldsymbol{s}}) \right\|_F, \tag{7}$$

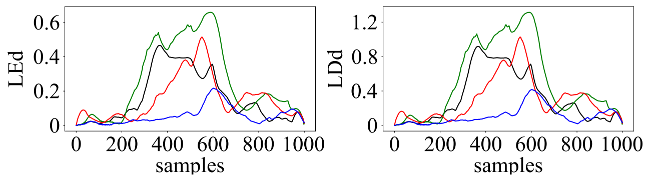


FIGURE 6. Distances of each pair of SPD matrix and quaternion.

where $\boldsymbol{s}$ and $\hat{\boldsymbol{s}}$ are the given and generated SPD matrices respectively, logm$(\cdot)$ is the matrix logarithm, and $\|\cdot\|_F$ indicates the Frobenius Norm.

For UQs, we use the Log Quaternion distance (LQd):

$$\mathrm{LQd} = \left\| 2\mathrm{Log}_{\boldsymbol{g}}(\langle \boldsymbol{q}, \bar{\hat{\boldsymbol{q}}} \rangle) \right\|, \tag{8}$$

where $\boldsymbol{q}$ is the given quaternion, $\bar{\hat{\boldsymbol{q}}}$ is the conjugate of the generated quaternion, and $\langle \cdot, \cdot \rangle$ represents the Hamilton product, whose result is a quaternion mapped to the tangent space by the (3) to be a vector.

For each demonstration, we also visualized the change of the LEd and LQd along the sequence in Fig. 6. From which we notice that the distance error in the middle of the set are larger than that of the start and the end. The reason is that the 4 starting samples are used to generated the whole 4 sets, and because of the stability of the model, in the end, they will converge to the last sample. Hence, the distances in the start and the end are very small. In the middle of the motion, the generated data depend on the pattern that the model has learned, and the error also accumulates there, so the distance reaches its highest in the middle. It is worth mentioning that the shape of the 2 graphs in in Fig. 6 is the same because they all come from the same data on the tangent space.

D. GENERALITY TESTING

The top 3 hyperparameter combinations, found using the approach described in the previous section, were computed considering only data of the shape "N". In this section, we investigate whether they are also proper for training models for other shapes. Therefore, we took the combination with the best performance to train the other shapes and compared the results with those from random combination but also with 11 layers (making sure that the 2 models have the same capacity).

We experimentally found out that the average DTW error ($\mathrm{DTW_e}$) of the whole dataset given by the best hyperparameter combination of "N" shape (mean $\mathrm{DTW_e} = 284$) is slightly better than that by random combination (mean $\mathrm{DTW_e} = 294$). Although it is not what we expected but it is also reasonable since the patterns to be learned by the model are distinguishing, the best combination for one pattern maybe just a random one for another pattern. As qualitatively shown in Fig. 7, where we visualized two sample motions on their manifolds, the learning process and the successive projection of the generated profiles on the corresponding RM is already accurate. Indeed the projection's analytical form is

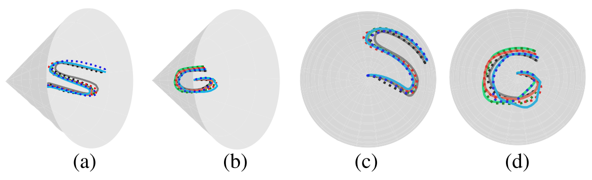


FIGURE 7. Two different shapes visualized on the respective RMs, i.e., $\mathcal{S}_2^+$ ((a) and (b)) and $\mathcal{S}^3$ ((c) and (d)). Light solid (dark dotted) lines represent the generated (demonstrated) data.

know and does not affect the accuracy. If the particular task demands for higher reproduction accuracy, it is recommended to search for a better combination of hyperparameters in the specific case. For instance, we observed in the case of "N" shape that the DTW error decreases from 378.5 to 274, i.e., by 27.6 % with the best set of hyperparameters.

Next, we also want to know the generality of the model concerning the demonstration area i.e., another perspective of the robustness of the learning results. To have a intuitive view of the model, we plotted the stream figures in the cubes containing the normalized data of demonstrations on the tangent space shown in Fig. 8. We plotted the generated trajectories, the corresponding demonstrations, and the stream lines (cyan curves) representing the velocity field (direction to the next predicted point). To provide a better visualization without all stream curves in the area messing together, we just selected the plane most fitting those demonstrations to show the shape patterns that the model has learned.

From these stream figures, the learning results of the model are well illustrated. First, it is clear that the models are able to accurately learn the patterns of very different shapes. Second, except the area that the demonstrations go through, the model also equips the surrounding space with the similar basic pattern which also proves the robustness of the model. Last, all the generated stream curves tend to converge to the final point of the demonstrations indicating the stability of the model.

E. COMPARISON

We compare our approach with Riemannian GMM (R-GMM) [24] and Fast Diffeomorphic Matching (FDM) [32], as well as with two baseline approaches. R-GMM extends the classic GMM/Gaussian Mixture Regression (GMR) appraoch to perform regression on UQ and SPD manifolds. FDM has been proposed to learn a diffeomorphism between Euclidean spaces. The procedure presented in Sec. IV allows to extend FDM to learn stable Riemannian skills. The baselines are naive versions of our procedure where we ignore the constraints from the manifold during the learning and post-process the generated data in order to fulfill the geometric constraints. In particular, unit quaternions are considered as 4-D Euclidean vectors during the learning and normalized after the generation. SPD matrices are vectorized by stacking the 3 unique components of the 2×2 SPD matrix into a

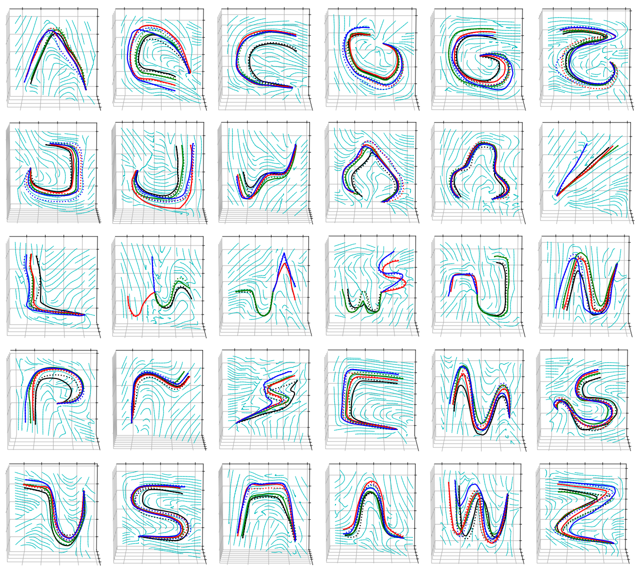


FIGURE 8. Stream figures of the demonstration area for each shape.

3-D Euclidean vector used for learning.[2] From the generated vector, we build a symmetric matrix and find the nearest SPD matrix to this one using the approach in [33]. For all the approaches, we train a separate model for each manifold (UQ and SPD) and each motion in the Riemannian LASA dataset (Sec. V-A). For validation, we take only the first point in each demonstration and use it to generate the entire trajectory of 1000 steps as described in Algorihm 1 (point 3). This procedure is commonly adopted to validate LfD approaches based on stable dynamical systems [5], [26], [32].

From the comparison results shown in Fig. 9, transferring the data to the tangent space and acquiring the corresponding model will lead better average performance (8.2% for SPD and 17.7% for UQ on average). The bottom row in Fig. 9 shows mean accuracy and standard deviation for each approach on the entire dataset. On average, RiemannianFlow outperforms all the considered approaches. Moreover, compared to the naive version, RiemannianFlow has similar performance, but more accurate as it has smaller mean and standard deviation. These results for SPD matrices are in-line with a previous study [4], where authors have shown that the approximation error becomes significant for higher dimensional matrices. Moreover, looking at the shapes 27 to 30 in Fig. 9 (first row), R-GMM and FDM fail in learning *multimodal* skills with different behaviors in different parts of the manifold (see the third row, columns 2 to 5 of Fig. 8). This is because FDM learns from a single (average) demonstration, while R-GMM takes a time-like input and regress always the same motion on the manifold. On the contrary, RiemannianFlow is capable to accurately encode single- and multimodal Riemannian skills.

[2]As SPD matrices are symmetric matrices, then we use Mandel's notation to vectorize the SPD matrices, and use the inverse of Mandel's notation to matricize an Euclidean vector.

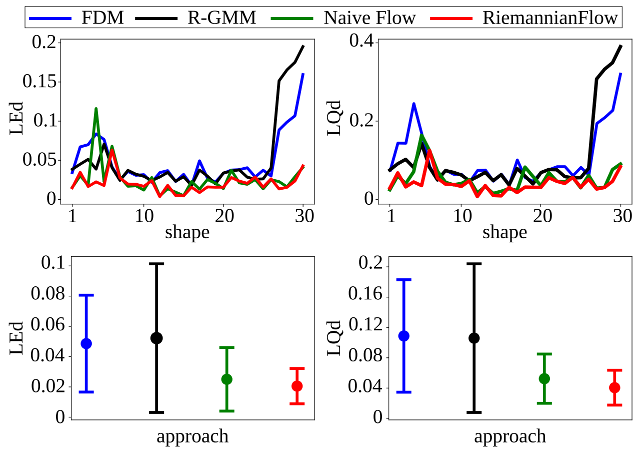


FIGURE 9. Comparative results obtained with different approaches on the Riemannian LASA dataset for SPD matrix (left column) and UQ (right column).

F. ROBOT EXPERIMENT

In order to evaluate our RiemannianFlow approach experimentally, we studied a typical industrial insertion task, namely Peg-in-Hole. However, we applied it to a shape fitting toy. In this game, the robot needs to follow accurately the demonstrated trajectory to perform a successful insertion. It is worth mentioning that, even if several state-of-the-art methods exist to solve PiH, we used PiH as an industrial application example as it shows the key features of our procedure.

We provided 7 kinesthetic demonstrations starting at different poses and converging to the hole pose (see Fig. 10) defined by the position of $\boldsymbol{g}_{\mathrm{p}} = [0.675, -0.051, 0.203]^{\top}$ m and the orientation of UQ: $\boldsymbol{g}_{\mathrm{q}} = 0.022 + [0.991, -0.125, 0.035]^{\top}$. The demonstrations are in the form of $\{\{\boldsymbol{p}^{demo}_{m,n}, \boldsymbol{q}^{demo}_{m,n}\}_{m=1}^{M}\}_{n=1}^{N}$ where $M = 4000$ is the total length of the demonstrations and $N = 7$ is the number of demonstrations. Robot Operating System (ROS) was used to record the demonstration data. Afterwards, we projected the UQ trajectories, from all demonstrations, to the tangent space $\mathcal{T}_{\boldsymbol{g}}\mathcal{S}^3$ using (3). Subsequently, we trained two models for both position and orientation trajectories. These 2 models were used later to predict a new pose trajectory that started from a new arbitrary pose, which was different from the demonstrations starting poses, and ended up in the hole pose performing a successful PiH insertion using ROS with the generated prediction data. An illustration of the demonstrations, the prediction (black dashed line), and robot pose (red solid line) trajectories when performing in the tangent space is shown in Fig. 11.

In this experiment we did not performed an hyperparameter search and used the combination of (11 layers, ReLu, Adam, 0.001) with random uniform initialization. From Fig. 11, although there were situations where the generated trajectories were not perfectly following the given demonstrations, the convergence of the final state was ensured and the task was successfully executed. Moreover, when the robot performed the task according to a new generated trajectory,

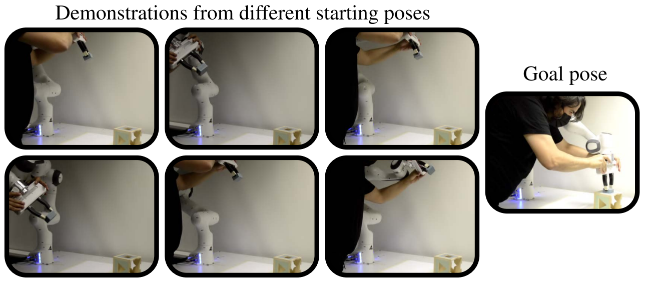


FIGURE 10. An illustration of the PiH experiment. A human operator is teaching the robot to perform PiH task by starting from different poses and ending up in the hole pose.

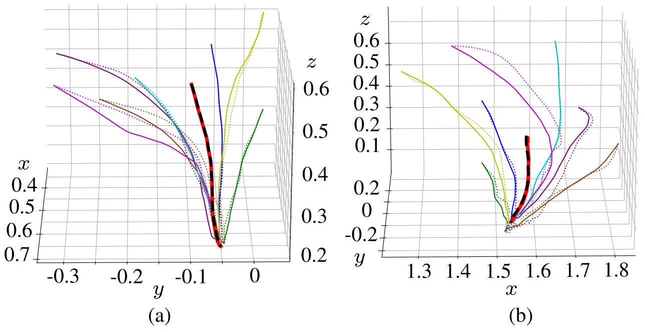


FIGURE 11. Results for the PiH task. (a) Position and (b) orientation (projected on the tangent space) data of the real robot experiment. The dash black curves represent the prediction for a new starting point, and the thick red lines indicate the movement that robot performs in both figures.

it also reached the same goal as expected. More executions of the task under different conditions is shown in the accompanying video.

VI. DISCUSSION

Experiments in Sec. V-E and V-F show the effectiveness of RiemannianFlow both in simulations and in a real experiment. The experimental comparison, performed on a public benchmark, has confirmed known results and highlighted some new findings. An expected result is that approaches that effectively learn from multiple demonstrations are in general more accurate. This can be clearly seen in the bottom plots of Fig. 9 where both Naive and RiemannianFlow outperform FDM and R-GMM. Another expected results is data normalization affects the accuracy, especially if the motion is integrated over time as the error accumulate. In our comparison, each motion lasts for 1000 steps and the error is still limited (although already visible in Fig. 9). It is clear that, in a real setting where the motion may last for several minutes, the accumulated error may cause the task to fail. Similar considerations apply to the Cholesky decomposition, where increasing the matrix size also affects accuracy [4]. Finally, from our comparison, it is possible to conclude that imitation learning approaches based on dynamical systems are sufficiently flexible to accurately encode complex manifold motions while preserving the stability.

The accuracy of RiemannianFlow comes at the cost of a longer training time (Sec. V-C), but in most cases, training can be performed in about 1 hour on a PC equipped with a consumer GPU—in our test where we used a NVIDIA RTX 2060 and acquire a good model. As for the hyperparameter search, an elaborate search will bring large gain of performance but cost longer time. And the search needs to be repeated to learn motions that significantly differ from each other (Sec. V-D). Once the network is trained, the time needed to generate one sample allows to close a control loop at about 50 Hz (Sec. V-C), which is sufficient for a kinematic controller. Therefore, RiemannianFlow can be potentially deployed on resource-constrained robots like mobile platforms, exploiting low-power GPUs like the NVIDIA Jetson series.

VII. CONCLUSION AND FUTURE WORK

In this paper, have we proposed a deep generative model to properly encode complex data that have to fulfil specific geometric constraints. From the geometric perspective, we consider those constrained data forming RMs, and utilize distance reserving mappings to project them on the tangent space, successfully releasing the constrains. After the learning in the tangent space, we project back the generated data on the manifold to re-fulfill the original constraints.

The resulting approach, namely RiemannianFlow, has been evaluated on a benchmark of Riemannian motions and in a PiH task with a real robot. Obtained results show that the approach has the ability of learning various kinds of complex Riemannian patterns while guaranteeing the stability and fulfilling geometric constraints for both stiffness (SPD matrices) and orientation data (UQs). Although the learned model is already accurate if the learning process starts with a random initialization of the hyperparameters, hyperparameters search has been shown to be an effective solution to significantly reduce the reproduction error, which is necessary for for tasks that require higher accuracy (e.g., PiH task). Last, through stream figures, we prove the high robustness of the model in a region around the demonstration area.

In the future work, we will try to let the model learn more control information including position, orientation, stiffness, and velocity at the same time, increasing the dependence among those attributes to handle more sophisticated situations. Further, we will find some faster initialization methods to give better performance in general.

ACKNOWLEDGMENT

Matteo Saveriano was with the Department of Computer Science, University of Innsbruck, 6020 Innsbruck, Austria.

REFERENCES

[1] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard, "Recent advances in robot learning from demonstration," *Annu. Rev. Control, Robot., Auto. Syst.*, vol. 3, pp. 297–330, May 2020.

[2] J. Vidakovic, B. Jerbic, B. Sekoranja, M. Svaco, and F. Suligoj, "Accelerating robot trajectory learning for stochastic tasks," *IEEE Access*, vol. 8, pp. 71993–72006, 2020.

[3] S. Xu, Y. Ou, J. Duan, X. Wu, W. Feng, and M. Liu, "Robot trajectory tracking control using learning from demonstration method," *Neurocomputing*, vol. 338, pp. 249–261, Apr. 2019.

[4] F. J. Abu-Dakka, L. Rozo, and D. G. Caldwell, "Force-based variable impedance learning for robotic manipulation," *Robot. Auto. Syst.*, vol. 109, pp. 156–167, Nov. 2018.

[5] J. Urain, M. Ginesi, D. Tateo, and J. Peters, "ImitationFlow: Learning deep stable stochastic dynamic systems by normalizing flows," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2020, pp. 5231–5237.

[6] J. Urain, D. Tateo, and J. Peters, "Learning stable vector fields on lie groups," 2021, *arXiv:2110.11774*.

[7] X. Chen, N. Wang, H. Cheng, and C. Yang, "Neural learning enhanced variable admittance control for human–robot collaboration," *IEEE Access*, vol. 8, pp. 25727–25737, 2020.

[8] L. Peternel, N. Tsagarakis, D. Caldwell, and A. Ajoudani, "Robot adaptation to human physical fatigue in human–robot co-manipulation," *Auto. Robots*, vol. 42, no. 5, pp. 1011–1021, Jun. 2018.

[9] K. Kronander and A. Billard, "Learning compliant manipulation through kinesthetic and tactile human–robot interaction," *IEEE Trans. Haptics*, vol. 7, no. 3, pp. 367–380, Jul. 2014.

[10] Y. Huang, L. Rozo, J. Silvério, and D. G. Caldwell, "Kernelized movement primitives," *Int. J. Robot. Res.*, vol. 38, no. 7, pp. 833–852, Jun. 2019.

[11] F. J. Abu-Dakka, Y. Huang, J. Silvério, and V. Kyrki, "A probabilistic framework for learning geometry-based robot manipulation skills," *Robot. Auto. Syst.*, vol. 141, Jul. 2021, Art. no. 103761.

[12] N. Jaquier, L. Rozo, D. G. Caldwell, and S. Calinon, "Geometry-aware manipulability learning, tracking, and transfer," *Int. J. Robot. Res.*, vol. 40, nos. 2–3, pp. 624–650, Feb. 2021.

[13] J. Silverio, L. Rozo, S. Calinon, and D. G. Caldwell, "Learning bimanual end-effector poses from demonstrations using task-parameterized dynamical systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2015, pp. 464–470.

[14] A. Ude, B. Nemec, T. Petric, and J. Morimoto, "Orientation in Cartesian space dynamic movement primitives," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2014, pp. 2997–3004.

[15] F. J. Abu-Dakka, B. Nemec, J. A. Jørgensen, T. R. Savarimuthu, N. Krüger, and A. Ude, "Adaptation of manipulation skills in physical contact with the environment to reference force profiles," *Auto. Robots*, vol. 39, no. 2, pp. 199–217, Aug. 2015.

[16] M. Saveriano, F. J. Abu-Dakka, A. Kramberger, and L. Peternel, "Dynamic movement primitives in robotics: A tutorial survey," 2021, *arXiv:2102.03861*.

[17] M. Saveriano, F. Franzel, and D. Lee, "Merging position and orientation motion primitives," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 7041–7047.

[18] F. J. Abu-Dakka, M. Saveriano, and L. Peternel, "Periodic DMP formulation for quaternion trajectories," in *Proc. 20th Int. Conf. Adv. Robot. (ICAR)*, Dec. 2021, pp. 658–663.

[19] S. Kim, R. Haschke, and H. Ritter, "Gaussian mixture model for 3-DoF orientations," *Robot. Auto. Syst.*, vol. 87, pp. 28–37, Jan. 2017.

[20] M. J. A. Zeestraten, I. Havoutis, J. Silvério, S. Calinon, and D. G. Caldwell, "An approach for imitation learning on Riemannian manifolds," *IEEE Robot. Autom. Lett.*, vol. 2, no. 3, pp. 1240–1247, Jul. 2017.

[21] L. D. Rozo and V. Dave, "Orientation probabilistic movement primitives on Riemannian manifolds," in *Proc. 5th Conf. Robot Learn.*, 2021, pp. 373–383.

[22] Y. Huang, F. J. Abu-Dakka, J. Silvério, and D. G. Caldwell, "Toward orientation learning and adaptation in Cartesian space," *IEEE Trans. Robot.*, vol. 37, no. 1, pp. 82–98, Feb. 2021.

[23] X. Pennec, "Manifold-valued image processing with SPD matrices," in *Riemannian Geometric Statistics in Medical Image Analysis*, X. Pennec, S. Sommer, and T. Fletcher, Eds. New York, NY, USA: Academic Press, 2020, pp. 75–134.

[24] S. Calinon, "Gaussians on Riemannian manifolds: Applications for robot learning and adaptive control," *IEEE Robot. Autom. Mag.*, vol. 27, no. 2, pp. 33–45, Jun. 2020.

[25] D. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1530–1538.

[26] S. M. Khansari-Zadeh and A. Billard, "Learning stable nonlinear dynamical systems with Gaussian mixture models," *IEEE Trans. Robot.*, vol. 27, no. 5, pp. 943–957, Oct. 2011.

[27] M. Müller, "Dynamic time warping," in *Information Retrieval for Music and Motion*. Berlin, Germany: Springer, 2007, pp. 69–84.

[28] C. F. Jekel, G. Venter, M. P. Venter, N. Stander, and R. T. Haftka, "Similarity measures for identifying material parameters from hysteresis loops using inverse analysis," *Int. J. Mater. Forming*, vol. 12, no. 3, pp. 355–378, May 2019.

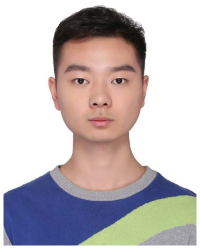
[29] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1026–1034.

[30] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, 2019, pp. 2623–2631.

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[32] N. Perrin and P. Schlehuber-Caissier, "Fast diffeomorphic matching to learn globally asymptotically stable nonlinear dynamical systems," *Syst. Control Lett.*, vol. 96, pp. 51–59, Oct. 2016.

[33] N. J. Higham, "Computing a nearest symmetric positive semidefinite matrix," *Linear Algebra Appl.*, vol. 103, pp. 103–118, May 1988.



WEITAO WANG received the B.S. degree in automation engineering from the University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 2020, and the master's degree from the Autonomous System Program, European Institute of Innovation and Technology (EIT), a joint program between KTH Royal Institute of Technology, Stockholm, Sweden, and Aalto University, Espoo, Finland, in 2022. Besides, he has been working as a Research Assistant at the Intelligent Robotics Group, EEA, Aalto University, since June 2021.

MATTEO SAVERIANO (Member, IEEE) received the B.Sc. and M.Sc. degrees in automatic control engineering from the University of Naples, Italy, in 2008 and 2011, respectively, and the Ph.D. degree from the Technical University of Munich, in 2017. Currently, he is an Assistant Professor at the Department of Industrial Engineering (DII), University of Trento, Italy. Previously, he was an Assistant Professor at the University of Innsbruck and a Postdoctoral Researcher at the German Aerospace Center (DLR). His research interests include robot learning, human–robot interaction, and understanding and interpreting human activities. He is an Associate Editor of IEEE Robotics and Automation Letters (webpage: https://matteosaveriano.weebly.com/).

FARES J. ABU-DAKKA (Member, IEEE) received the B.Sc. degree in mechanical engineering from Birzeit University, Palestine, in 2003, and the M.Sc. and Ph.D. degrees in robotics motion planning from the Polytechnic University of Valencia, Spain, in 2006 and 2011, respectively. Currently, he is a Senior Researcher at the Intelligent Robotics Group, EEA, Aalto University, Finland. Previously, he was researching at ADVR, Istituto Italiano di Tecnologia (IIT). From 2013 to 2016, he was holding a Visiting Professor position at ISA, Carlos III University of Madrid, Spain. His research interests include robot learning and control and human–robot interaction. He is an Associate Editor of ICRA, IROS, and IEEE Robotics and Automation Letters (webpage: https://sites.google.com/view/abudakka/).