
Article:
Wu, S. orcid.org/0009-0001-9266-0799, Wang, Y. and Huang, Y. orcid.org/0000-0002-5395-5076 (2025) AutoLfD: Closing the Loop for Learning from Demonstrations. IEEE Transactions on Automation Science and Engineering. ISSN 1545-5955

https://doi.org/10.1109/tase.2025.3532820

AutoLfD: Closing the Loop for Learning from Demonstrations

Shaokang Wu, Yijin Wang, and Yanlong Huang*

***Abstract*—Over the past few years, there have been numerous works towards advancing the generalization capability of robots, among which learning from demonstrations (LfD) has drawn much attention by virtue of its user-friendly and data-efficient nature. While many LfD solutions have been reported, a key question has not been properly addressed: how can we evaluate the generalization performance of LfD? For instance, when a robot draws a letter that needs to pass through new desired points, how does it ensure the new trajectory maintains a similar shape to the demonstration? This question becomes more relevant when a new task is significantly far from the demonstrated region. To tackle this issue, a user often resorts to manual tuning of the hyperparameters of an LfD approach until a satisfactory trajectory is attained. In this paper, we aim to provide closed-loop evaluative feedback for LfD and optimize LfD in an automatic fashion. Specifically, we consider dynamical movement primitives (DMP) and kernelized movement primitives (KMP) as examples and develop a generic optimization framework capable of measuring the generalization performance of DMP and KMP and auto-optimizing their hyperparameters. Evaluations including peg-in-hole, block-stacking and pushing tasks on a real robot evidence the applicability of our framework.**

***Note to Practitioners*—The paper is motivated by the demand to transfer human skills to robots. While the problems of 'what to learn' and 'how to learn' have been long-standing research topics, the solutions for evaluating the quality of such skill transfer remain largely open. We introduce a novel closed-loop framework towards transferring human skills to robots in an automatic manner. Specifically, we collect a training dataset that reflects user preference for trajectory adaptation and train a trajectory encoder network using the dataset. With the encoder network, we design a robust metric to measure the skill transfer quality and subsequently employ the metric to guide imitation learning of human skills. By using our framework, unseen robotic tasks can be tackled by adapting the demonstrations straightforwardly, where relevant hyperparameters involved in skill transfer are optimized automatically.**

***Index Terms*—Learning from demonstrations, generalization metric, trajectory encoder network, dynamical movement primitives, kernelized movement primitives.**

I. Introduction

Learning from demonstrations (LfD), as a long-standing research topic in the scope of robot learning, aims to endow robots with the capability of mimicking human behaviours from a single or a few pre-provided demonstrations [1]–[3]. Notably, additional policy updates via interaction with the environment or human users can be incorporated into a LfD paradigm.

All authors are with the School of Computer Science, University of Leeds, Leeds LS29JT, UK. (scswu@leeds.ac.uk; ml20y3w@leeds.ac.uk; y.l.huang@leeds.ac.uk).
∗ Corresponding author

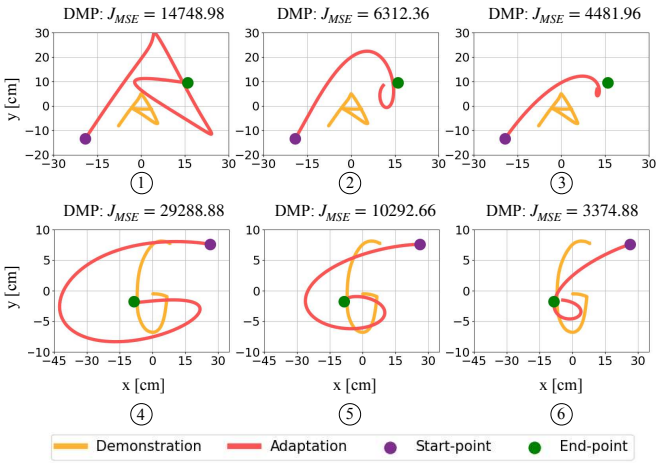

Fig. 1: The writing of 2-D letters using DMP, where DMP is employed to imitate the demonstration and generate a new trajectory starting from a new starting point towards a new target point. The forcing term in DMP is learned by GP. Each new trajectory (i.e., adaptation) for letters 'A' and 'G' corresponds to a different set of hyperparameters for DMP. J_{MSE} is defined as the MSE loss between the adapted trajectory and the demonstration.

There is a large body of literature in LfD that has been dedicated to addressing two key problems: (*i*) what to learn and (*ii*) how to learn. For the first problem, various works (e.g., [4]–[10]) studied the learning of different forms of demonstrations depending on task requirements, e.g., Cartesian position and velocity, orientation and angular velocity, joint position and velocity, stiffness and damping matrices, manipulability, interaction forces, as well as various combinations of these profiles. For the second problem, many LfD solutions have been proposed under different assumptions about the learning model. For instance, by encoding a demonstration as a spring-damper model with an additional forcing term modulating the acceleration profile, dynamical movement primitives (DMP) were developed [11]. Task-parameterized Gaussian mixture model (TP-GMM) was proposed in [12], where consistent features underlying demonstrations within each task frame were modelled by GMM and later these features, together with new task frames, were utilized to deal with task generalization. In [13], a demonstration was assumed to be the weighted sum of a series of basis functions and multivariate Gaussian distribution was used to model the distribution of multiple demonstrations, leading to a probabilistic LfD framework, i.e., probabilistic movement primitives (ProMP). In contrast to ProMP which maximizes the likelihood of demonstrations, we considered the posterior instead and introduced a non-parametric LfD solution

[14], i.e., kernelized movement primitives (KMP), where the explicit definition of basis functions was mitigated.

While the community witnesses remarkable progress in LfD in terms of algorithms and applications, one crucial question seems to be overlooked: how to evaluate the generalization performance of LfD? Indeed, in supervised learning and reinforcement learning settings, the evaluation of generalization is straightforward, e.g., using mean squared error (MSE) for regression, cross-entropy loss (CEL) for classification, and reward for reinforcement learning. However, the design of a sensible metric for LfD is nontrivial.

Considering the task of writing a 2-D letter (see Fig. 1) and assuming that only a single demonstration of this letter is available, if we use DMP to write it from a new start-point towards a new end-point, how can we ensure the new trajectory is smooth enough and meanwhile maintains the shape of the demonstration? Note that the adapted trajectory must be different from the demonstration as a consequence of the new start-point and end-point demands. In this case, the traditional treatment of using MSE or maximum a posterior (MAP) [14] as a generalization metric could become problematic, especially when the adapted trajectory is far from the demonstration, see Fig. 1 where the best adaptation has the largest MSE loss.

Similar limitations of MSE apply to other distance-aware metrics as well, including Fréchet distance [15] and dynamic time warping (DTW) [16]. Often, we rely on manual tuning of the hyperparameters of DMP until a proper trajectory meeting our requirements is obtained. The tuning process heavily relies on our experience and usually involves many trials, which restrains the deployment of DMP in dynamic environments since such tuning is required whenever a new task requirement arises. A similar issue of tuning hyperparameters is also encountered in other LfD solutions, including ProMP and KMP.

In this paper, we propose an automatic optimization framework for LfD (i.e., *autoLfD*) to free users from the laborious tuning of hyperparameters, where a novel metric capable of measuring the generalization performance of any LfD approach is designed. Such a metric is built on a trajectory encoder network that is trained on a dataset comprising manually labelled positive and negative adaptation samples. Specifically, we take DMP and KMP as examples and use this metric to guide the optimization of their hyperparameters. We begin with preliminaries on DMP and KMP in Section II. After that, we discuss the limitations of MSE and MAP acting as generalization metrics for LfD (Section III). In Section IV, we present a novel metric for evaluating the generalization performance of LfD and explain hyperparameters optimization for DMP and KMP using the metric, where both gradient descent (GD) and Bayesian optimization (BO) are exploited. We test the autoLfD framework (Section V) in several scenarios including simulated writing tasks, as well as peg-in-hole, block-stacking, and pushing tasks implemented in a real robot. We discuss relevant work in Section VI and the limitations of autoLfD in Section VII. We conclude this work in Section VIII.

II. Preliminaries

In this section, we briefly review the basic rationale of DMP (Section II-A) and KMP (Section II-B), which will be later optimized using the proposed autoLfD framework.

A. DMP

DMP, consisting of a first-order canonical system and a second-order transformation system, can learn and generalize the motion pattern underlying a single demonstration. Formally, DMP encodes trajectories as [11]

$$\begin{aligned} &\tau\dot{s} = -\alpha s, \\ &\tau^2\ddot{\boldsymbol{\xi}} = \mathbf{K}_p(\mathbf{g}-\boldsymbol{\xi}) - \tau\mathbf{K}_v\dot{\boldsymbol{\xi}} + s(\mathbf{g}-\boldsymbol{\xi}_0)\odot\mathbf{f}_{\mathbf{w}}(s), \end{aligned} \tag{1}$$

where $\mathbf{K}_p$ and $\mathbf{K}_v$ are stiffness and damping matrices. τ, α, and s denote motion duration, decay factor, and phase variable, respectively. $\boldsymbol{\xi} \in \mathbb{R}^{\mathcal{O}}$ represents $\mathcal{O}$-dimensional position while $\dot{\boldsymbol{\xi}}$ and $\ddot{\boldsymbol{\xi}}$ respectively denote the corresponding velocity and acceleration. $\boldsymbol{\xi}_0$ is the starting point and $\mathbf{g}$ is the target. $\odot$ stands for the element-wise product. $\mathbf{f}_{\mathbf{w}}(s) \in \mathbb{R}^{\mathcal{O}}$ represents the forcing term driven by s, where the parameter vector $\mathbf{w}$ is learned from the demonstration.

In contrast to the classical representation of approximating the forcing term $\mathbf{f}(s)$ as a linear combination of a set of predefined basis functions, Gaussian process (GP) was suggested in [17] to model $\mathbf{f}(s)$ from a nonparametric perspective, where the explicit definition of basis functions is mitigated and fewer open parameters are demanded.

Suppose we have access to a demonstration of time-length N, i.e., $\{t_n, \boldsymbol{\xi}_n, \dot{\boldsymbol{\xi}}_n, \ddot{\boldsymbol{\xi}}_n\}_{n=1}^N$, we can substitute the demonstration into (1) and extract a new training dataset $\{s_n, \mathbf{f}(s_n)\}_{n=1}^N$. The new dataset can be learned by GP to predict the corresponding forcing term $\mathbf{f}(s)$ for an arbitrary $s \in (0, 1]$. Given an inquiry s^*, we have [18]

$$f_i(s^*) = \mathbf{k}^*(\mathbf{K}+\lambda\mathbf{I})^{-1}\mathbf{U}_i, \tag{2}$$

with

$$\begin{aligned} &\mathbf{k}^* = [k(s^*, s_1)\ k(s^*, s_2)\ \cdots\ k(s^*, s_N)], \\ &\mathbf{K}_{i,j} = k(s_i, s_j), \\ &\mathbf{U}_i = [f_i(s_1)\ f_i(s_2)\ \cdots\ f_i(s_N)]^\top, \end{aligned}$$

where $k(\cdot,\cdot)$ denotes a kernel function, e.g., the definition of a commonly used squared exponential (SE) kernel is $k(s_i, s_j) = \exp(-k_h(s_i - s_j)^2)$ with a hyperparameter $k_h > 0$. $\mathbf{k}^* \in \mathbb{R}^{1\times N}$, $\mathbf{U}_i \in \mathbb{R}^N$, $\mathbf{K}_{i,j}$ denotes the element at the i-th row and the j-th column of the matrix $\mathbf{K} \in \mathbb{R}^{N\times N}$. $f_i(\cdot)$ corresponds to the i-th element of $\mathbf{f}(\cdot)$. $\lambda > 0$ is a scalar, and $\mathbf{I}$ is an identity matrix.

B. KMP

Unlike DMP which learns a single demonstration, KMP learns the probabilistic distribution of multiple demonstrations. Given H demonstrations $\{\{t_{n,h}, \boldsymbol{\xi}_{n,h}, \dot{\boldsymbol{\xi}}_{n,h}\}_{n=1}^N\}_{h=1}^H$ with $\boldsymbol{\xi}_{n,h}$ being the trajectory point at the n-th time step from the h-th demonstration, their distribution can be modelled by GMM and Gaussian mixture regression (GMR) [12], [19],

leading to a probabilistic reference trajectory that encapsulates the distribution of demonstrations, i.e., $\{t_n, \hat{\boldsymbol{\mu}}_n, \hat{\boldsymbol{\Sigma}}_n\}_{n=1}^N$, where $\mathcal{P}\left(\begin{bmatrix}\boldsymbol{\xi}_n \\ \dot{\boldsymbol{\xi}}_n\end{bmatrix} | t_n\right) = \mathcal{N}(\hat{\boldsymbol{\mu}}_n, \hat{\boldsymbol{\Sigma}}_n)$.

Let us denote $\hat{\boldsymbol{\mu}} = [\hat{\boldsymbol{\mu}}_1^\top \ \hat{\boldsymbol{\mu}}_2^\top \ \cdots \ \hat{\boldsymbol{\mu}}_N^\top]^\top$ and $\hat{\boldsymbol{\Sigma}} = \text{blockdiag}(\hat{\boldsymbol{\Sigma}}_1, \hat{\boldsymbol{\Sigma}}_2, \ldots, \hat{\boldsymbol{\Sigma}}_N)$. For a query input t^*, KMP predicts its corresponding output as [14]

$$\boldsymbol{\mu}(t^*) = \begin{bmatrix}\boldsymbol{\xi}(t^*) \\ \dot{\boldsymbol{\xi}}(t^*)\end{bmatrix} = \mathbf{k}^*(\mathbf{K} + \lambda\hat{\boldsymbol{\Sigma}})^{-1}\hat{\boldsymbol{\mu}}, \tag{3}$$

where

$$\mathbf{k}^* = [\mathbf{k}(t^*, t_1) \ \mathbf{k}(t^*, t_2) \ \cdots \ \mathbf{k}(t^*, t_N)],$$
$$\mathbf{K}_{i,j} = \mathbf{k}(t_i, t_j).$$

Both $\mathbf{k}^*$ and $\mathbf{K}$ depend on the extended kernel matrix $\mathbf{k}(\cdot,\cdot)$ whose elements are $k(\cdot,\cdot)$ and the associated first-order and second-order derivatives, see [14] for more details. $\lambda > 0$ is used to mitigate the overfitting issue.

Note that the predictions in (2) and (3) have a nonparametric form, and proper hyperparameter tuning is needed, including the kernel parameter k_h involved in the kernel function $k(\cdot,\cdot)$ and the regularization factor λ. Throughout the paper, we will use the notation $\boldsymbol{\Theta} = [k_h \ \lambda]^\top$ to represent the collection of hyperparameters to be optimized.

III. Motivation

Why do we need a novel metric to measure the generalization performance of LfD? In order to answer this question, we first formulate the generalization (i.e., adaptation) problem in LfD (Section III-A), and subsequently present some examples to evidence the issues arising from the distance-aware metrics MSE and MAP (Section III-B). Here, we take MSE and MAP as motivation examples, more evaluations on Fréchet distance and DTW are reported in Section V.

A. Problem formulation

For the sake of brevity, we rewrite the demonstration of DMP as $\hat{\boldsymbol{\mu}} = [\hat{\boldsymbol{\mu}}_1^\top \ \hat{\boldsymbol{\mu}}_2^\top \ \cdots \ \hat{\boldsymbol{\mu}}_N^\top]^\top$, where $\hat{\boldsymbol{\mu}}_n = [\hat{\boldsymbol{\xi}}_n^\top \ \dot{\hat{\boldsymbol{\xi}}}_n^\top]^\top$. Note that the same notation $\hat{\boldsymbol{\mu}}_n$ is also used to denote the mean of the reference trajectory in KMP, which should be straightforwardly distinguished from the context. Given a demonstration $\{t_n, \hat{\boldsymbol{\mu}}_n\}_{n=1}^N$ for DMP or a probabilistic reference trajectory $\{t_n, \hat{\boldsymbol{\mu}}_n, \hat{\boldsymbol{\Sigma}}_n\}_{n=1}^N$ for KMP, as well as adaption constraints $\mathbf{c}$ (e.g., desired positions of new start-point, via-point, and end-point), we can generate an adapted trajectory as $\boldsymbol{\mu}^* = [\boldsymbol{\mu}_1^{*\top} \ \boldsymbol{\mu}_2^{*\top} \ \cdots \ \boldsymbol{\mu}_N^{*\top}]^\top$ via DMP or KMP, where each predicted datapoint $\boldsymbol{\mu}_n^* = [\boldsymbol{\xi}_n^{*\top} \ \dot{\boldsymbol{\xi}}_n^{*\top}]^\top \in \mathbb{R}^{2\mathcal{O}}$ comprises both position and velocity.

Our goal is to build an automatic optimization framework (which is referred to as autoLfD) for DMP and KMP to guide the optimization of their hyperparameters $\boldsymbol{\Theta}$. To do so, a metric $J(\hat{\boldsymbol{\mu}}, \boldsymbol{\mu}^*)$ that measures the discrepancy between the demonstration $\hat{\boldsymbol{\mu}}$ and the adapted trajectory $\boldsymbol{\mu}^*$ will be required. Moreover, such a metric should take trajectory smoothness into account. Suppose we have such a metric $J(\cdot,\cdot)$ at hand, we can view trajectory adaptation as a function

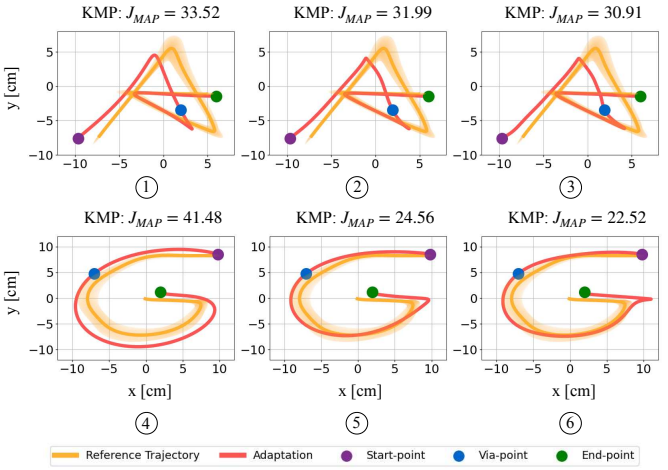

Fig. 2: Assessment of the generalization performance of KMP using the MAP metric. In ①–③, KMP is used to adapt the 2-D demonstration 'A' towards a new start-point, via-point, and end-point, where each adaptation corresponds to a different set of hyperparameters of KMP; similar applications of KMP in writing the letter 'G' are showcased in ④–⑥.

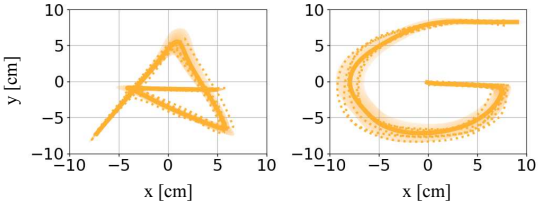

Fig. 3: Extracting the probabilistic reference trajectory from multiple demonstrations using GMR, where the yellow dotted curves denote demonstrations. The yellow solid curves and the shaded areas depict the means and covariances of the probabilistic reference trajectories, respectively.

$\boldsymbol{\mu}^* = \mathcal{G}_{\boldsymbol{\Theta}}(\hat{\boldsymbol{\mu}}, \mathbf{c})$ and formulate the hyperparameter optimization for LfD as minimizing $J(\hat{\boldsymbol{\mu}}, \boldsymbol{\mu}^*) = J(\hat{\boldsymbol{\mu}}, \mathcal{G}_{\boldsymbol{\Theta}}(\hat{\boldsymbol{\mu}}, \mathbf{c}))$. Here, constraints $\mathbf{c}$ represent $\mathcal{O}$-dimensional target point for DMP and $2\mathcal{O}$-dimensional desired start-/via-/end-point (comprising both position and velocity) for KMP.

B. Limitations of MSE and MAP

We use MSE as the generalization metric for DMP, since DMP learns a single demonstration. The MSE metric is

$$J_{\text{MSE}}(\hat{\boldsymbol{\mu}}, \boldsymbol{\mu}^*) = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{\mu}_n^* - \hat{\boldsymbol{\mu}}_n)^\top(\boldsymbol{\mu}_n^* - \hat{\boldsymbol{\mu}}_n). \tag{4}$$

As KMP essentially maximizes the posterior conditioned on the demonstrations, we use MAP as the generalization metric for KMP. As discussed in [14], the MAP metric for KMP constitutes part of the Kullback-Leibler (KL) divergence loss. Formally, we have the MAP metric as $\prod_{n=1}^N \mathcal{P}(\boldsymbol{\mu}_n^* | \hat{\boldsymbol{\mu}}_n, \hat{\boldsymbol{\Sigma}}_n)$. With the definition of multivariate Gaussian distribution, the MAP metric amounts to the minimization of

$$J_{\text{MAP}} = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{\mu}_n^* - \hat{\boldsymbol{\mu}}_n)^\top\hat{\boldsymbol{\Sigma}}_n^{-1}(\boldsymbol{\mu}_n^* - \hat{\boldsymbol{\mu}}_n). \tag{5}$$

In the following discussion, we refer to (5) as a MAP cost.

The writing examples using DMP under different hyperparamters are provided in Fig. 1. In plots ①–③, the adapted trajectories (shown as red curves) for the letter 'A' become distorted when the MSE cost decreases, showing that the MSE

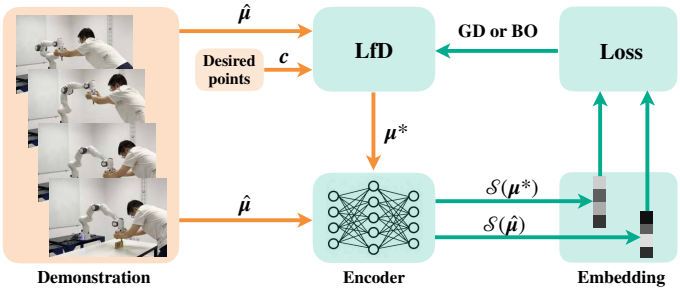

Fig. 4: An overview of the autoLfD framework, where the hyperparameters of an LfD approach are constantly optimized towards reducing the distance between the feature vectors $\mathcal{S}(\hat{\boldsymbol{\mu}})$ and $\mathcal{S}(\boldsymbol{\mu}^*)$. Here, $\hat{\boldsymbol{\mu}}$ and $\boldsymbol{\mu}^*$ represent the demonstration and the adapted trajectory generated by LfD, respectively.

cost fails to indicate the shape maintenance. Similarly, MSE is unable to measure the adaptation performance of DMP in writing the letter 'G' (see plots ④–⑥), where the adapted trajectory that resembles the shape of the demonstration and reaches the new target precisely, however, has the largest MSE cost (see ④).

Similarly, we evaluate the MAP costs on the trajectories generated by KMP in Fig. 2, where a desired via-point is imposed for either letter in addition to the desired new start-point and end-point. The probabilistic reference trajectories for 'A' and 'G' are extracted from their corresponding five demonstrations via GMR [12], [14], as shown in Fig. 3. As with the observations in Fig. 2, the best adaptations (plotted by red curves) have the largest MAP costs (see ① for 'A' and ④ for 'G'), while the adapted trajectories with significant distortions have smaller MAP costs, see ③ and ⑥.

From the above examples, we can conclude that optimizing the hyperparameters of DMP with the MSE metric and KMP with the MAP metric could be problematic, since both metrics focus on the 'reproduction' of the demonstration or the reference trajectory (i.e., staying close to the demonstration or the reference trajectory in terms of Euclidean distance) without considering the motion shape and smoothness requirements, thus failing to provide a reliable indicator of the generalization performance of LfD methods.

IV. AutoLfD

We have discussed the issues of using MSE and MAP as adaptation metrics, now we propose a novel framework autoLfD that allows for optimizing the hyperparameters of DMP and KMP automatically, where an encoder network that transforms trajectories into a latent feature space is designed (Section IV-A). With the encoder network, the generalization metric can be designed naturally (Section IV-B1). After that, we discuss two routes to hyperparameter optimization, including GD (Section IV-B2) and BO (Section IV-B3). An overview of the proposed autoLfD framework is provided in Fig. 4.

A. Trajectory encoder network

The encoder network takes an entire trajectory (i.e., concatenation of all trajectory points) as input, rather than a single point. Given a demonstration $\hat{\boldsymbol{\mu}}$ and an LfD approach $\mathcal{G}$, we randomly sample desired points $\mathbf{c}$ and LfD hyperparameters $\boldsymbol{\Theta}$, and subsequently obtain the adapted trajectory $\boldsymbol{\mu}^* = \mathcal{G}_{\boldsymbol{\Theta}}(\hat{\boldsymbol{\mu}}, \mathbf{c})$. If the adaptation is smooth enough and meanwhile maintains the shape of the demonstration, we will label it as a positive training sample $\mathbf{p}$; otherwise, it will be labeled as a negative sample $\mathbf{n}$. By repeating such a procedure, we can collect a training dataset to encapsulate various demonstrations and associated adaptations under different hyperparameters for the LfD method.

Algorithm 1: *Trajectory encoder network*

```
repeat
    Sample a batch of triplets from the dataset;
    Calculate the triplet loss with (6);
    Update the encoder via backpropagation;
until encoder parameters converge;
```

We train the encoder network using triplets in the form of $< \hat{\boldsymbol{\mu}}, \mathbf{p}, \mathbf{n} >$, where $\mathbf{p}$ and $\mathbf{n}$ respectively represent satisfactory (i.e., 'positive' samples) and unsatisfactory (i.e., 'negative' samples) adaptation in comparison with the learned demonstration $\hat{\boldsymbol{\mu}}$. Learning both positive and negative samples differs from previous LfD methods that focus on learning positive trajectories – this treatment helps to capture user preference for generalization similarity better. Formally, given a batch comprising M triplets, we minimize the triplet loss [20]

$$\sum_{m=1}^{M} \max\big(0, \gamma + ||\mathcal{S}(\hat{\boldsymbol{\mu}}_m) - \mathcal{S}(\mathbf{p}_m)||_2 - ||\mathcal{S}(\hat{\boldsymbol{\mu}}_m) - \mathcal{S}(\mathbf{n}_m)||_2\big), \tag{6}$$

where $\mathcal{S}(\cdot) : \mathbb{R}^{2\mathcal{O}N} \rightarrow \mathbb{R}^h$ corresponds to the encoding of a trajectory into an h-dimensional feature vector, $\|\cdot\|_2$ represents ℓ_2 norm. The constant margin $\gamma > 0$ ensures that the positive and negative samples can be more easily distinguished. The triplet loss shares the same spirit as the Siamese networks [21], [22], which have been proven effective in many applications, e.g., face recognition and signature verification. The procedure of training the encoder network is summarized in Algorithm 1.

B. An automatic optimization framework for LfD

1) Generalization metric : Following the spirit of the loss function in (6) – 'positive' trajectories should stay close to the demonstration while 'negative' trajectories should stay away from the demonstration, we can define a metric to measure the distance (i.e., dissimilarity) between a demonstration $\hat{\boldsymbol{\mu}}$ and an adapted trajectory $\boldsymbol{\mu}^*$ under condition $\mathbf{c}$, i.e.,

$$\mathcal{L}_{\boldsymbol{\Theta}} = ||\mathcal{S}(\hat{\boldsymbol{\mu}}) - \mathcal{S}(\boldsymbol{\mu}^*)||_2 = ||\mathcal{S}(\hat{\boldsymbol{\mu}}) - \mathcal{S}(\mathcal{G}_{\boldsymbol{\Theta}}(\hat{\boldsymbol{\mu}}, \mathbf{c}))||_2. \tag{7}$$

With the optimal hyperparameters $\boldsymbol{\Theta}$ that minimizes $\mathcal{L}_{\boldsymbol{\Theta}}$, both DMP and KMP are able to generate adapted trajectories that mostly resemble the demonstration. Note that the metric in (7) operates at the level of trajectories, which can be combined with many other LfD methods beyond DMP and KMP.

2) Gradient descent : To search for the optimal $\boldsymbol{\Theta}$ minimizing $\mathcal{L}_{\boldsymbol{\Theta}}$ in (7), a common optimization technique is a gradient-based method, i.e., GD. Specifically, GD iteratively updates $\boldsymbol{\Theta}$ via $\boldsymbol{\Theta} := \boldsymbol{\Theta} - \eta \nabla_{\boldsymbol{\Theta}} \mathcal{L}_{\boldsymbol{\Theta}}$, where $\eta > 0$ is the learning

Algorithm 2: *AutoLfD*

1 **Initialization:**
2 Given a demonstration (or reference trajectory) $\hat{\boldsymbol{\mu}}$, desired points $\mathbf{c}$ ▷ e.g., start-point/via-point/end-point;
3 Select an LfD approach $\mathcal{G}$ ▷ e.g., DMP and KMP;
4 Train the trajectory encoder network with Algorithm 1;
5 **Update LfD hyperparamters:**
6 Initialize the hyperparameters $\boldsymbol{\Theta}$ of DMP/KMP;
7 Generate an adapted trajectory $\boldsymbol{\mu}^* = \mathcal{G}_{\boldsymbol{\Theta}}(\hat{\boldsymbol{\mu}}, \mathbf{c})$ using DMP/KMP with the current hyperparameters $\boldsymbol{\Theta}$;
8 Calculate the dissimilarity between $\boldsymbol{\mu}^*$ and $\hat{\boldsymbol{\mu}}$ via (7) ▷ Replacing (7) with other metrics leads to baselines;
9 Update $\boldsymbol{\Theta}$ using GD or BO;
10 Repeat line 7–9 until $\boldsymbol{\Theta}$ converges.

rate. $\nabla_{\boldsymbol{\Theta}}\mathcal{L}_{\boldsymbol{\Theta}}$ stands for the gradient of (7) with respect to $\boldsymbol{\Theta}$ and is computed by chain rule, i.e.,

$$\nabla_{\boldsymbol{\Theta}}\mathcal{L}_{\boldsymbol{\Theta}} = \nabla_{\mathcal{S}(\boldsymbol{\mu}^*)}\mathcal{L}\big(\mathcal{S}(\hat{\boldsymbol{\mu}}), \mathcal{S}(\boldsymbol{\mu}^*)\big)\nabla_{\boldsymbol{\mu}^*}\mathcal{S}(\boldsymbol{\mu}^*)\nabla_{\boldsymbol{\Theta}}\mathcal{G}_{\boldsymbol{\Theta}}(\hat{\boldsymbol{\mu}}, \mathbf{c}), \tag{8}$$

where

$$\nabla_{\mathcal{S}(\boldsymbol{\mu}^*)}\mathcal{L}(\mathcal{S}(\hat{\boldsymbol{\mu}}), \mathcal{S}(\boldsymbol{\mu}^*)) = \frac{\mathcal{S}(\boldsymbol{\mu}^*) - \mathcal{S}(\hat{\boldsymbol{\mu}})}{\|\mathcal{S}(\hat{\boldsymbol{\mu}}) - \mathcal{S}(\boldsymbol{\mu}^*)\|_2},$$
$$\boldsymbol{\mu}^* = [\boldsymbol{\mu}_1^{*\top} \ \ldots \ \boldsymbol{\mu}_N^{*\top}]^\top = \mathcal{G}_{\boldsymbol{\Theta}}(\hat{\boldsymbol{\mu}}, \mathbf{c}).$$

We minimize the loss function $\mathcal{L}_{\boldsymbol{\Theta}}$ using PyTorch, a modern deep learning framework that automatically computes the gradients $\nabla_{\boldsymbol{\Theta}}\mathcal{G}_{\boldsymbol{\Theta}}(\hat{\boldsymbol{\mu}}, \mathbf{c})$ and $\nabla_{\boldsymbol{\mu}^*}\mathcal{S}(\boldsymbol{\mu}^*)$ via backpropagation.

3) Bayesian optimization : In contrast to GD which relies on explicit calculation of gradient and could converge to a local minimum, BO is a gradient-free, global optimization method. BO resorts to an acquisition function to decide the query points for evaluations towards finding the optimal input point with a minimal number of searching steps, where the core idea is to balance the exploitation and exploration when sampling queries.

We take one of the most popular acquisition functions expected improvement (EI) [23], [24] as an example. Suppose we have evaluated a set of hyperparameters $\{\boldsymbol{\Theta}_1, \boldsymbol{\Theta}_2, \ldots, \boldsymbol{\Theta}_n\}$ and obtained their corresponding metric costs $\{\mathcal{L}_{\boldsymbol{\Theta}_1}, \mathcal{L}_{\boldsymbol{\Theta}_2}, \ldots, \mathcal{L}_{\boldsymbol{\Theta}_n}\}$ via (7), the next query point $\boldsymbol{\Theta}_{n+1}$ is determined by

$$\begin{aligned}\boldsymbol{\Theta}_{n+1} &= \underset{\boldsymbol{\Theta}}{\operatorname{argmax}}\ \mathbb{E}_{\mathcal{L}_{\boldsymbol{\Theta}}} \max\big(\mathcal{L}^* - \mathcal{L}_{\boldsymbol{\Theta}}, 0\big) \\ &= \underset{\boldsymbol{\Theta}}{\operatorname{argmax}} \Big\{\varphi\big(\frac{\mathcal{L}^* - \mu_{\mathcal{L}|\mathcal{D}}}{K_{\mathcal{L}|\mathcal{D}}}\big)K_{\mathcal{L}|\mathcal{D}} \\ &\qquad + (\mathcal{L}^* - \mu_{\mathcal{L}|\mathcal{D}})\Phi\big(\frac{\mathcal{L}^* - \mu_{\mathcal{L}|\mathcal{D}}}{K_{\mathcal{L}|\mathcal{D}}}\big)\Big\}.\end{aligned} \tag{9}$$

Here, the expectation is estimated over the distribution $\mathcal{L}_{\boldsymbol{\Theta}} \sim \mathcal{GP}(\mu_{\mathcal{L}|\mathcal{D}}, K_{\mathcal{L}|\mathcal{D}})$, which is predicted by GP at the input $\boldsymbol{\Theta}$ given observations $\mathcal{D} = \{\boldsymbol{\Theta}_i, \mathcal{L}_{\boldsymbol{\Theta}_i}\}_{i=1}^n$. $\mathcal{L}^*$ is the smallest one among $\{\mathcal{L}_{\boldsymbol{\Theta}_i}\}_{i=1}^n$. $\varphi(\cdot)$ and $\Phi(\cdot)$ represent the multivariate normal distribution and its cumulative distribution, respectively. Once the new query point $\boldsymbol{\Theta}_{n+1}$ is known and evaluated using (7), a new pair $\{\boldsymbol{\Theta}_{n+1}, \mathcal{L}_{\boldsymbol{\Theta}_{n+1}}\}$ will be added to the existing observations $\mathcal{D}$, and subsequently the search strategy in (9) will be employed again to decide the following query point $\boldsymbol{\Theta}_{n+2}$. By repeating the above procedure, the optimal $\boldsymbol{\Theta}^*$ associated with the minimal metric loss $\mathcal{L}_{\boldsymbol{\Theta}^*}$ will be found.

TABLE I: Training Hyperparameters of the Trajectory Encoder Network in Writing Tasks

Hyperparameters	**Values**
Learning rate	10^{-7}
Batch size	200
Epoch	30000
Margin	0.5
Dataset size	3026

Given a demonstration (or a reference trajectory) $\hat{\boldsymbol{\mu}}$, adaptation constraints $\mathbf{c}$, and hyperparameters $\boldsymbol{\Theta}$, we can use DMP or KMP to generate an adapted trajectory $\boldsymbol{\mu}^*$ that addresses the imitation of the demonstration and the adaptation constraints (i.e., desired points). Subsequently, we can measure the dissimilarity between the demonstration and the adapted trajectory by comparing their feature vectors $\mathcal{S}(\hat{\boldsymbol{\mu}})$ and $\mathcal{S}(\boldsymbol{\mu}^*)$ (encoded by the trajectory encoder network) using the metric in (7). The hyperparameters of DMP or KMP can be optimized via GD or BO towards reducing the loss (7). Once the optimization converges, the adapted trajectory from DMP or KMP using the optimal hyperparameters $\boldsymbol{\Theta}^*$ will be a proper generalization (in terms of shape maintenance and trajectory smoothness) from the demonstration $\hat{\boldsymbol{\mu}}$ under the constraints $\mathbf{c}$.

The entire autoLfD optimization framework is summarized in Algorithm 2. Note that the trajectory encoder is trained with the triplet loss (6) beforehand and its parameters are kept fixed when using autoLfD. If we replace the metric (7) in Algorithm 2 with MSE, MAP, Fréchet distance or DTW metric, we can obtain the baselines utilized in our experiments.

V. Evaluations

In this section, we report the evaluations of autoLfD in simulated letter-writing tasks (Section V-A), including a comprehensive comparison with several baselines (i.e., MSE, MAP, Fréchet distance, DTW). We also assess autoLfD in three real robotic tasks: peg-in-hole, block-stacking, and pushing tasks (Section V-B). Specifically, we answer the following questions in the evaluations:

(i) Is the metric in (7) a proper indicator of the generalization performance?
(ii) How does our metric perform against other metrics (i.e. MSE, MAP, Fréchet distance, and DTW)?
(iii) Which optimization algorithm between GD and BO can achieve superior performance?

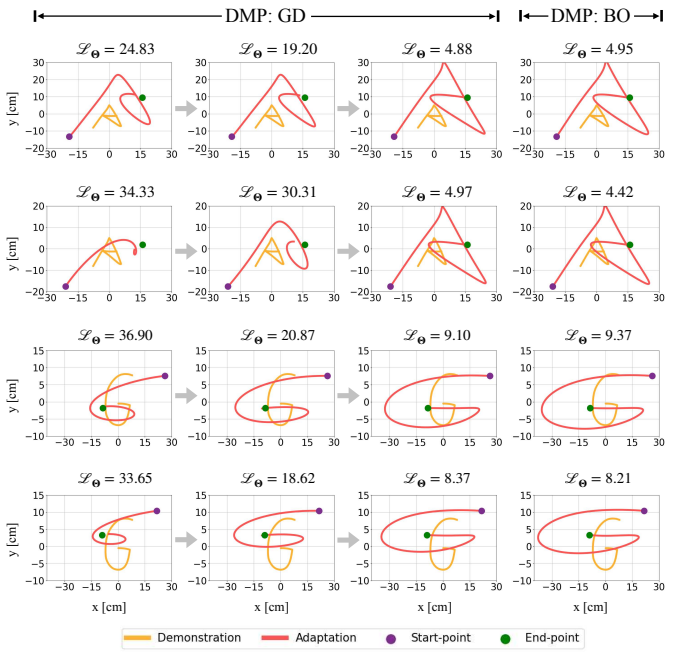

Fig. 5: Evaluations of autoDMP in letter-writing tasks. The first, second and third columns correspond to the updated hyperparameters of DMP via GD, where the adapted trajectories (with the grey arrows denoting the update process of GD) are gradually improved in terms of reaching the desired end-point and resembling the shape of the demonstration. The fourth column shows the adapted trajectories with the optimal hyperparameters found by BO. It is worth noting that, the desired points in the first and second rows are set differently. The third and fourth rows also have different desired points.

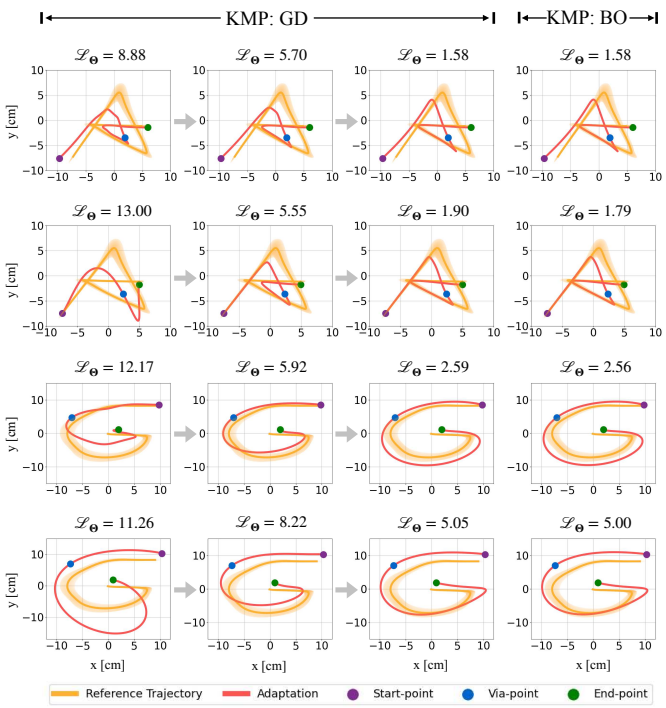

Fig. 6: Evaluations of autoKMP on letter-writing tasks, where both GD and BO are implemented. The grey arrows illustrate the update process of GD. The probabilistic reference trajectory for either letter is extracted from five demonstrations using GMM and GMR. For either letter, two groups of desired points are used.

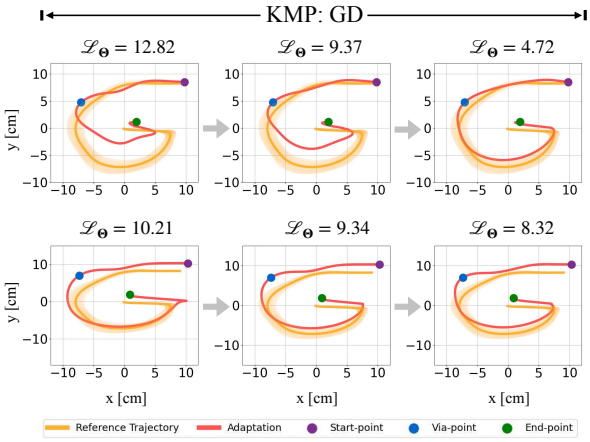

Fig. 7: Evaluations of autoKMP with GD optimization, where the hyperparameters of KMP are initialized with inappropriate values. The top and bottom rows correspond to different settings of desired points.

A. *Letter-writing tasks*

Since the demonstrations and corresponding adaptations have the same length, we train a unified encoder for evaluating adaptations, irrespective of the LfD approach and the demonstration letter to be learned. Table I lists the relevant hyperparameters involved in training the trajectory encoder network. We employ GD and BO to optimize the hyperparameters of DMP and KMP towards reducing the metric cost in (7), respectively.

1) Optimizing the hyperparameters of DMP : We test the writing of letters 'A' and 'G' using our framework and DMP (i.e., *autoDMP*), where the letters are obtained from [25]. The adapted trajectories for both letters are plotted in Fig. 5. Take the first row as an example, the adapted trajectory (depicted by the red curve in the first plot) generated by DMP with an initial setting of hyperparameters fails to reach the desired end-point and maintain the shape of the demonstration (plotted by the yellow curve). After updating the hyperparameters $\boldsymbol{\Theta}$ of DMP via GD a few times, the adapted trajectory is improved in terms of the trajectory shape (see the second plot), but it is still unable to reach the desired target precisely. After 30 updates, the final trajectory (see the third figure) reaches the desired target while resembling the shape of the demonstration. Note that careful initialization of hyperparameters for GD is demanded in all evaluations in Fig. 5 since GD could be trapped into an inappropriate local minimum.

In Fig. 5, the desired points (i.e., start-point and end-point) in the first row are the same as the ones in ①–③ in Fig. 1, and the desired points in the third row are the same as the ones in ④–⑥ in Fig. 1. However, differing from the MSE metric, the proposed metric loss $\mathcal{L}_\Theta$ in (7) indeed decreases when the adapted trajectory becomes better. Similarly, in the second and fourth rows, the smaller the metric loss is, the better the adaptation is.

The adapted trajectories under the optimized hyperparameters via BO are plotted in Fig. 5, showing that BO can achieve satisfactory trajectories for different letters and adaptation conditions. Here, proper initialization is not required for BO since BO is essentially a sampling-based method that uses the acquisition function to direct the sampling process.

2) Optimizing the hyperparameters of KMP : In addition to the evaluations on DMP, we assess the performance of autoLfD on KMP (i.e., *autoKMP*) as well, see Fig. 6. Note

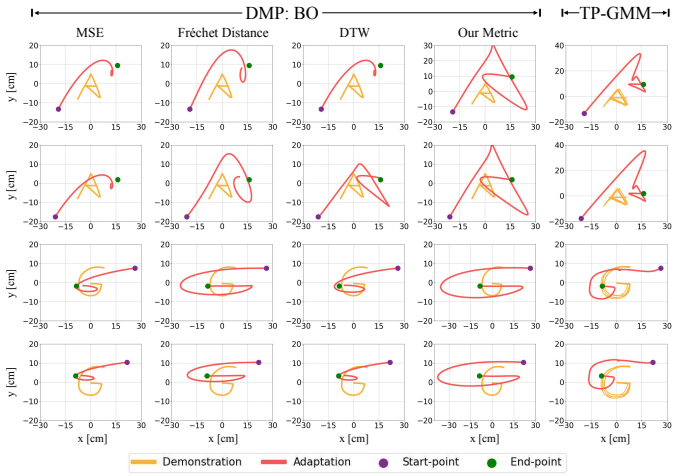

Fig. 8: Evaluations of MSE, Fréchet distance, DTW, and our metric on DMP, as well as TP-GMM. In each row, the same desired points are imposed.

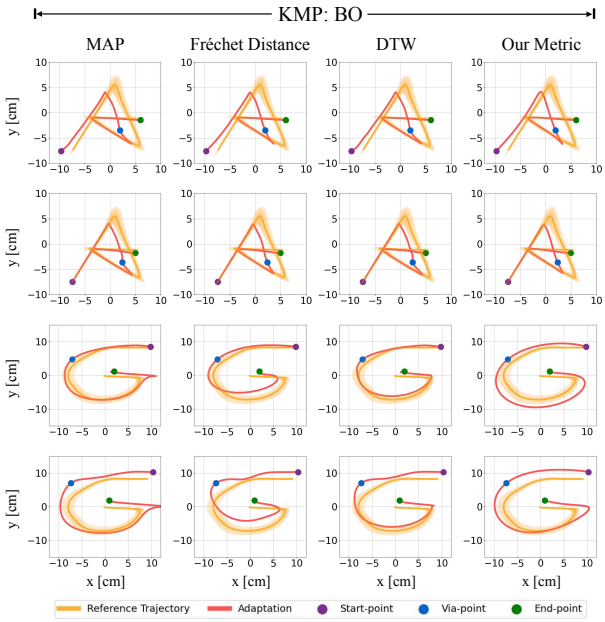

Fig. 9: Evaluations of MAP, Fréchet distance, DTW, and our metric on KMP. The same desired points are set in each row.

that the first and third rows of Fig. 6 have the same desired points as ①–③ and ④–⑥ in Fig. 2, respectively. Unlike MAP, our metric can indicate the adaptation performance properly, i.e., the generalization using KMP improves as the metric loss decreases. Moreover, by using either GD or BO, our framework can optimize the hyperparameters of KMP towards better generalization (i.e., in terms of shape and smoothness) under different adaptation conditions.

3) Comparison between GD and BO : While GD and BO yield similar performances in Fig. 5 and Fig. 6, there is an essential difference between them. In fact, GD is sensitive to the initial values of the hyperparameters. Given inappropriate initialization, GD could lead to undesired convergence, see examples of utilizing autoKMP in Fig. 7. Although the first row and the second row of Fig. 7 have the same desired points as the third and fourth rows of Fig. 6, the adapted trajectories in Fig. 7 either reach the desired end-point with an abrupt change or exhibit certain distortion. Compared with GD which may lead to undesired local minimum, BO is a global optimization technique that provides a more stable solution for the autoLfD framework. Therefore, we only adopt BO for the following evaluations.

Note that in the first and third rows of Fig. 5, the metric costs $\mathcal{L}_{\Theta}$ from BO (i.e., 4.95 for 'A' and 9.37 for 'G') are larger than the optimal costs from GD (i.e., 4.88 for 'A' and 9.10 for 'G') by neglectable margins, given that the margins only take a tiny proportion of the initial costs (i.e., 24.83 and 36.90). In fact, if we sample more samples in BO (100 iterations are used in Fig. 5), BO can lead to smaller costs, but the improvement of trajectories will be hardly observed for both letters.

4) Comparison with baselines : We now consider the same setup in Fig. 5 and compare our metric (defined in (7)) with MSE, Fréchet distance, and DTW metrics, where BO is utilized to optimize hyperparameters of DMP. For our metric, we use 100 iterations for BO. For each of the other metrics, we use 200 iterations. Considering that TP-GMM is designed to handle adaptations that take place in a region away from the demonstration region [12], [26], we implemented TP-GMM as a baseline as well.

The first three columns of Fig. 8 depict adaptations using MSE, Fréchet distance, and DTW metrics, respectively. The fifth column corresponds to TP-GMM, where five demonstrations are learned. We can see that all adaptations in these columns show certain distortions, and in some cases, the adaptation fails to reach the desired target, which is highly undesired. Note that the desired points in Fig. 8 force the adapted trajectories to stay far from the corresponding demonstrations. In this case, MSE, Fréchet distance and DTW become inappropriate because they are essentially distance-aware metrics and cannot gauge shape similarity between two trajectories. In the last column, the trajectory generated by TP-GMM has a significant change around the middle of the curve. The reason is that the new start-point and end-point in TP-GMM deviate from the demonstrations towards different directions with disproportionate offsets. In contrast, our metric ensures that DMP generates satisfactory generalization in different cases, see the fourth column of Fig. 8.

TABLE II: Planning errors and trajectory smoothness costs of DMP

Metric	**Target point error** (cm)	**Smoothness cost** (cm/s^2)	**Shape Maintenance**
MSE	4.19	$\mathbf{4.55 \times 10^2}$	No
Fréchet distance	3.21	1.15×10^3	No
DTW	3.06	7.16×10^2	No
Our metric	**0.35**	1.96×10^3	**Yes**

We calculate the planning errors (w.r.t. the desired target point) and smoothness costs of the optimized DMP trajectories under different metrics in Table II, where the smoothness cost is determined by $\frac{1}{N}\sum_{n=1}^{N}||\ddot{\boldsymbol{\xi}}_n||_2$. In terms of the planning error, our metric achieves the smallest error, resulting in a reduction of target error by 91.65%, 89.10%, and 88.56% compared to the MSE, Fréchet distance, and DTW metrics, respectively. The large planning errors for such metrics imply that they may fail to accomplish tasks relying on precise operations, e.g., picking an object at a specified location. Note that MSE, Fréchet distance and DTW metrics have smaller smoothness costs than our metric, whereas their correspond-

TABLE III: Planning errors and trajectory smoothness costs of KMP

Metric	Start-point error (cm)	Via-point error (cm)	End-point error (cm)	Smoothness cost (cm/s^2)	Shape Maintenance
MAP	$\mathbf{1.53 \times 10^{-4}}$	$\mathbf{7.30 \times 10^{-5}}$	$\mathbf{3.84 \times 10^{-4}}$	7.35×10^{2}	No
Fréchet Distance	2.11×10^{-4}	1.23×10^{-4}	4.45×10^{-4}	5.78×10^{2}	No
DTW	1.57×10^{-4}	8.36×10^{-5}	4.13×10^{-4}	6.91×10^{2}	No
Our metric	2.68×10^{-4}	1.94×10^{-4}	4.24×10^{-4}	$\mathbf{5.46 \times 10^{2}}$	**Yes**

ing trajectories can not maintain the demonstration's shape, therefore violating the primary goal of "imitation" learning.

Furthermore, we test MAP, Fréchet distance, and DTW metrics on KMP using the same setup in Fig. 6. Evaluations of these metrics are presented in Fig. 9, where our metric uses 100 iterations for BO and each of the other metrics uses 200 iterations. Here, TP-GMM is not evaluated as it can not deal with the via-point adaptation requirement. By observing Fig. 9, we can conclude that our metric exhibits more reliable performance across the writing tasks. The planning errors (w.r.t. various desired points) and the smoothness costs are reported in Table III. Using our metric, the smoothness cost is reduced by 25.71%, 5.54%, and 20.98% compared to the MAP, Fréchet distance, and DTW metrics, respectively. These results demonstrate that our metric achieves the best performance in terms of trajectory smoothness. Note that all metrics in fact have trivial planning errors at the magnitudes of 10^{-4} or $10^{-5}cm$ – the impact of such errors can be ignored in real robotic tasks. In addition, our metric is the only one that can maintain the shape of demonstrations, in contrast to other metrics.

The writing task was also studied in [27], where a sigmoidal decay phase variable was employed in DMP. The evaluations of [27] are given in the first three columns in Fig. 10, where the first, second, and third columns correspond to adaptations using 15, 30 and 50 basis functions to approximate the forcing term in DMP, respectively. While the first and second columns show certain distortions and(or) fail to reach the desired end-points, the third column shows proper accomplishment of different writing tasks. However, the satisfactory adaptations in the third column are obtained at the cost of time-consuming, repetitive, manual tuning of relevant parameters, including centres and bandwidths of basis functions, and parameters in the sigmoidal function. Indeed, building a training dataset for the trajectory encoder network before implementing autoDMP is also time-consuming. However, labelling positive and negative samples in 2-D writing tasks as per the user's preference could be trivial for a non-expert since only two classes are considered. Once the training dataset is obtained, the trajectory encoder network can be trained. After that, autoDMP can be deployed straightforwardly in similar but new tasks, enabling automatic optimization of relevant parameters for DMP (see the last column in Fig. 10). In contrast, tuning DMP parameters in [27] largely relies on the user's experience and it can be a challenging task for a non-expert. Moreover, whenever a new task is given, a new cycle of manually tuning parameters for DMP is required.

At the end of this section, we evaluate a hyperparameter optimization method in [28], where BO is used as the optimization tool and 1000 iterations are run. The trajectories generated by [28] are provided in the fourth column of Fig. 10. We can see that none of the optimized trajectories

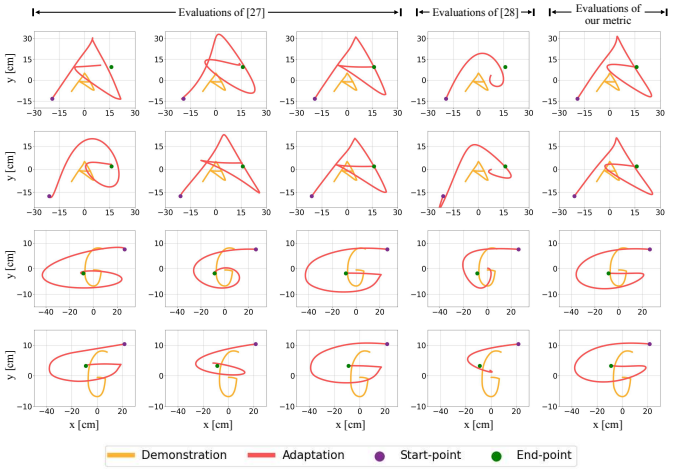

Fig. 10: Evaluations of a variant of DMP [27] and a parameter optimization method [28], as well as autoDMP. The first three columns correspond to different numbers of Gaussian basis functions. The last two columns show trajectories generated by [28] and autoDMP, respectively.

TABLE IV: Training Hyperparameters of the Trajectory Encoder Network in Real Robotic Tasks

Hyperparameters	Peg-in-Hole Task	Pushing Task	Block-Stacking Task
Learning rate	10^{-5}	10^{-5}	10^{-5}
Batch size	100	50	100
Epoch	20000	20000	15000
Margin	0.5	0.5	5
Dataset size	431	301	1416

maintains the shape of the corresponding demonstration. The main reason is that the objective function to be minimized in [28] only takes the reproduction error into account and ignores the shape maintenance requirement in letter-writing tasks.

B. *Real robot experiments*

So far, we have reported the performance of autoLfD in writing 2-D letters under various constraints of desired points, we now carry out real-world experiments using a seven-degree-of-freedom robot. Specifically, we assess the effectiveness of the autoLfD framework by performing a peg-in-hole task and a block-stacking task using DMP, and a pushing task using KMP. The relevant training hyperparameters are outlined in Table IV.

1) Peg-in-hole task : The goal of the peg-in-hole task is to insert a peg into a desired hole. An illustration of collecting a demonstration in such a task is shown in the first row of Fig. 11. Given that the diameter of the hole is slightly larger than that of the peg, it is crucial that the robot inserts the peg into the hole from a vertically downward direction when the peg is approaching the hole. Any deviation from a vertical insertion motion could result in misalignment between the peg and the hole, ultimately leading to unsuccessful tasks. Since this task demands a start-point (i.e., the initial position of the peg) and an end-point (i.e., the location of the hole), we employ autoDMP to generalize the demonstration to unseen new tasks.

We collect a single demonstration for the peg-in-hole task, as depicted by the grey curve in Fig. 12. To verify the effectiveness of autoDMP, we consider two settings: (*i*) a new

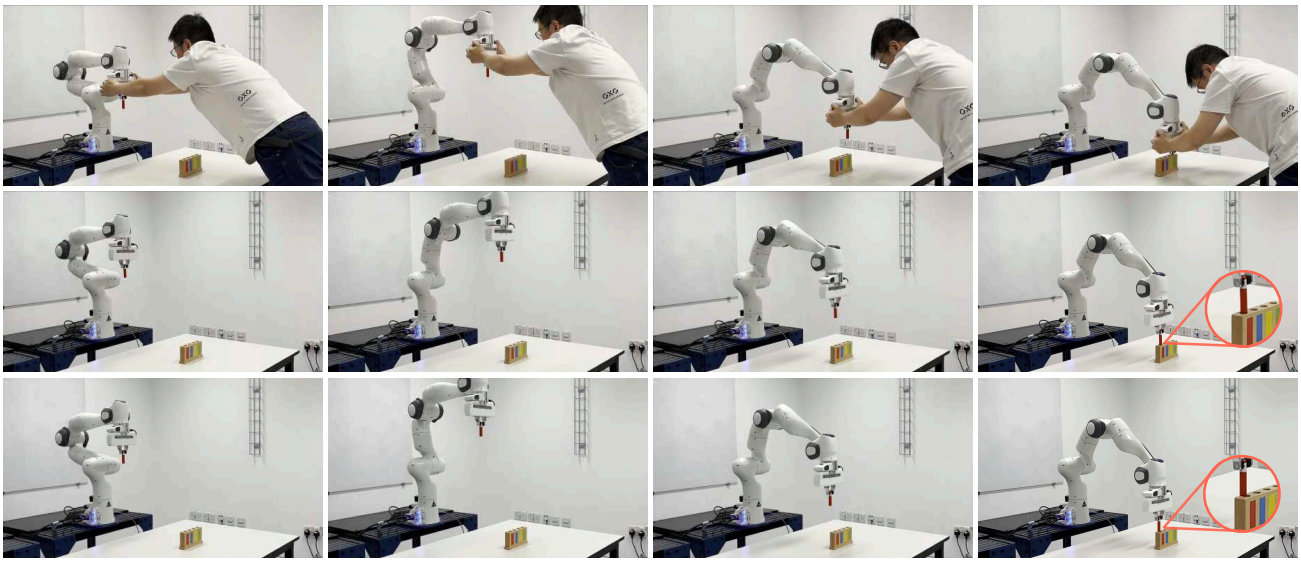

Fig. 11: Snapshots of the peg-in-hole task. *First row* shows the kinesthetic teaching of the peg-in-hole task. *Second and third rows* correspond to the adapted robot trajectories that are optimized using our metric in (7) and the MSE metric.

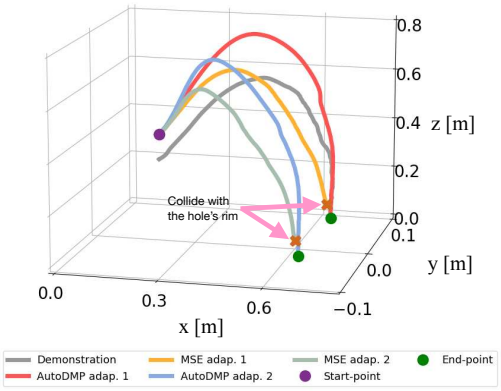

Fig. 12: The demonstration and adapted real robotic trajectories in the peg-in-hole task, where two adaptation settings are considered and in either setting our metric (i.e., autoDMP) and the MSE metric are implemented, respectively.

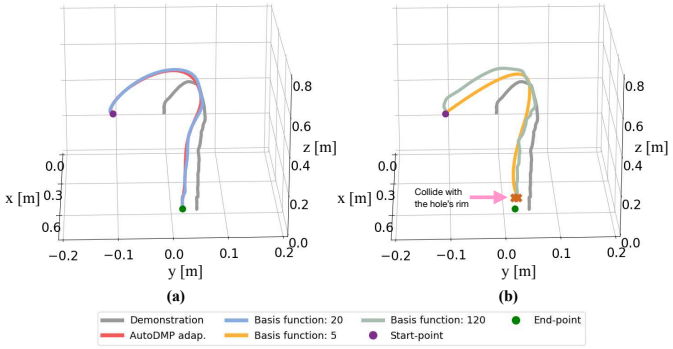

Fig. 13: Evaluations of autoDMP and a variant of DMP [29], where 5, 20, and 120 Gaussian basis functions are separately employed for [29].

start-point (see the purple dot in Fig. 12) that is away from the initial point of the demonstration and an end-point that is the same as that of the demonstration; (*ii*) a new start-point that is the same as the one used in (*i*) and a new end-point that is far from the target of the demonstration. As a comparison, we also implement hyperparameter updates for DMP using the MSE metric and BO in both settings. The number of iterations used in BO is set to 200 for autoDMP and MSE, respectively.

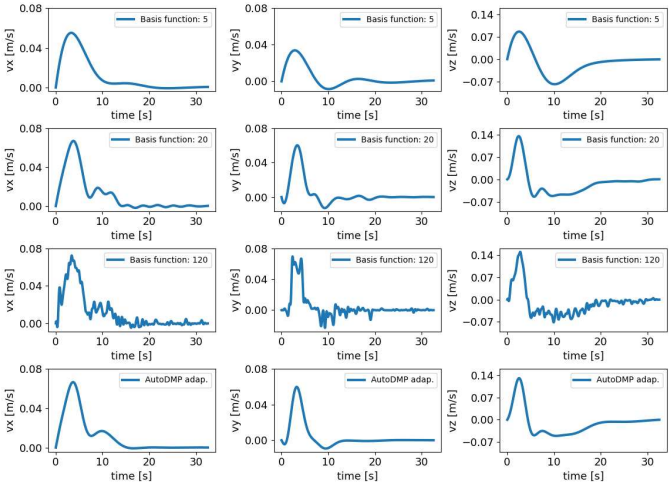

Fig. 14: The planned velocities via autoDMP and a variant of DMP [29].

TABLE V: Planning errors, trajectory smoothness costs and task completion status of DMP in peg-in-hole task

Metric	Target point error (m)	Smoothness cost (m/s^2)	Status	Shape Maintenance
Basis function: 5 [29]	4.16×10^{-3}	$\mathbf{1.14 \times 10^{-2}}$	Fail	No
Basis function: 20 [29]	1.12×10^{-3}	1.67×10^{-2}	**Success**	**Yes**
Basis function: 120 [29]	$\mathbf{9.28 \times 10^{-5}}$	4.76×10^{-2}	Fail	No
Our method	4.30×10^{-4}	1.47×10^{-2}	**Success**	**Yes**

The real robotic trajectories in both evaluation settings are plotted in Fig. 12, where we can see that the adapted trajectories with our metric exhibit vertical insertion motion near the desired end-points while the trajectories with the MSE metric approach the end-points from oblique directions. Snapshots of the experiments in the second evaluation setting are provided in Fig. 11, where the robot using the MSE metric indeed fails to insert the peg into the desired hole (see the third row of Fig. 11) as a consequence of the collision between the peg and the rim of the hole. In contrast, the robot can accomplish the peg-in-hole task successfully using autoDMP (see the second row of Fig. 11).

The peg-in-hole task was also investigated in [29], where

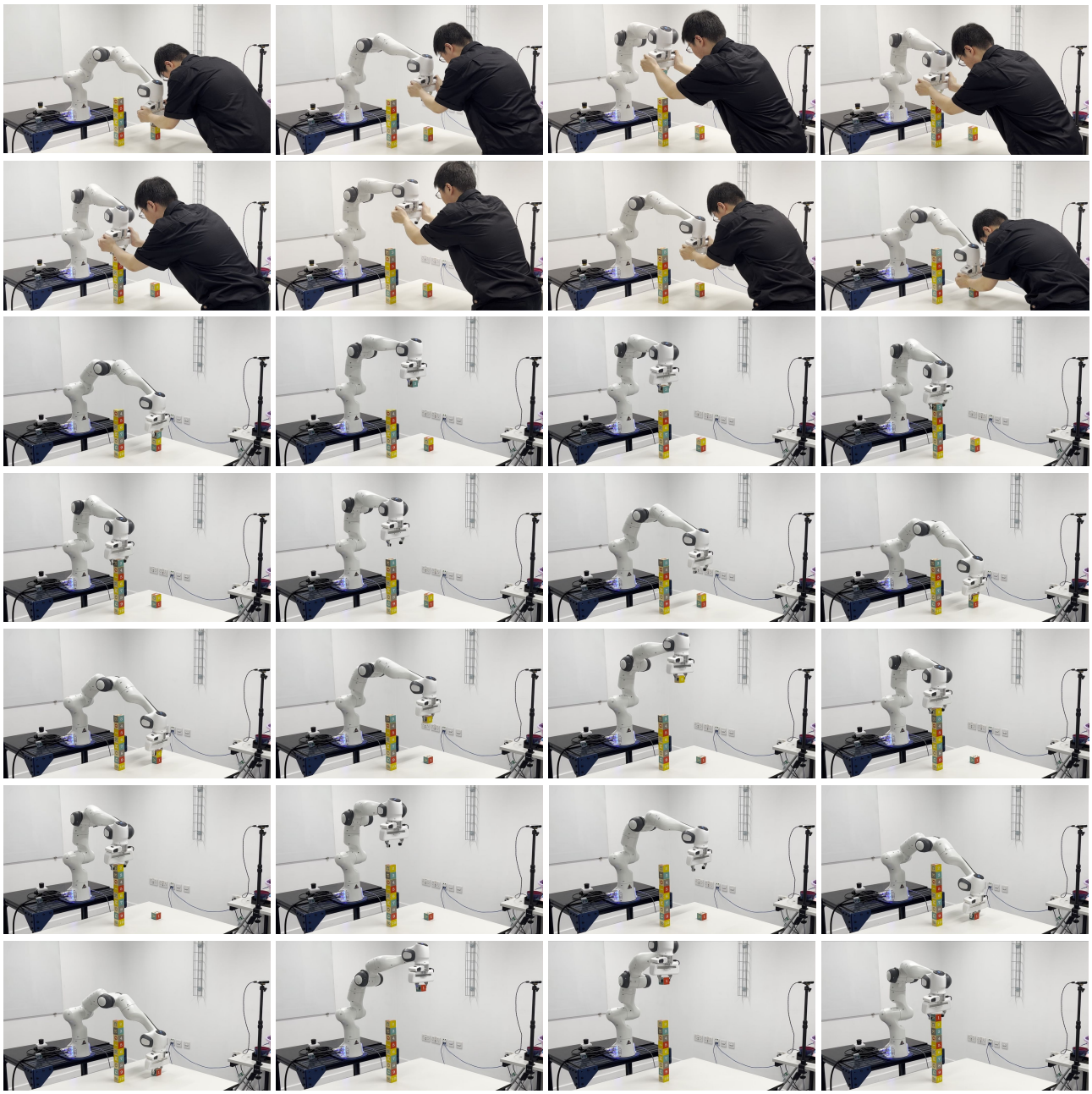

Fig. 15: Snapshot of the block-stacking task. The *first and second* rows show human demonstrations of the placing and picking tasks, respectively. The *third until seventh* rows show five subtasks and each subtask corresponds to either placing or picking a block.

DMP with frame relative goals was leveraged to tackle the task. We consider a new setting (*iii*) that is very different from previous settings (*i*) and (*ii*) to evaluate [29]. Specifically, the starting point in setting *(iii)* is away from the demonstration's initial point by 15*cm*. We have three evaluation groups with 5, 20 and 120 basis functions for [29], respectively. Note that [29] lacks a formal metric for assessing the generalization capability of DMP, we resort to manual tuning of relevant parameters for each group of evaluation.

By observing the real robotic trajectories plotted in Fig. 13, we can see that the adaptations associated with 5 and 120 basis functions fail to accomplish the peg-in-hole task due to the collision with the desired hole's rim. The planned velocities using [29] (see the first three rows of Fig. 14) demonstrate that, as the number of basis functions increases, the velocity curves transition from under-fitting (i.e., the first row) to over-fitting (see oscillations in the third row). Both under-fitting and over-fitting adaptations are unable to accomplish the task. A summary of [29] and our method is given in Table V. While using 20 basis functions for [29] succeeds, our method achieves smaller planning error (w.r.t. the target) and smaller smoothness cost (also see the last row of Fig. 14). Specifically, our method leads to 61.61% reduction in terms of target error and 11.98% reduction in terms of smoothness cost.

2) Block-stacking task: To further showcase the generalization capability of our approach, we consider a challenging long-horizon task – stacking blocks. The objective of this task is to repeatedly pick a block from a stack and place it on top of

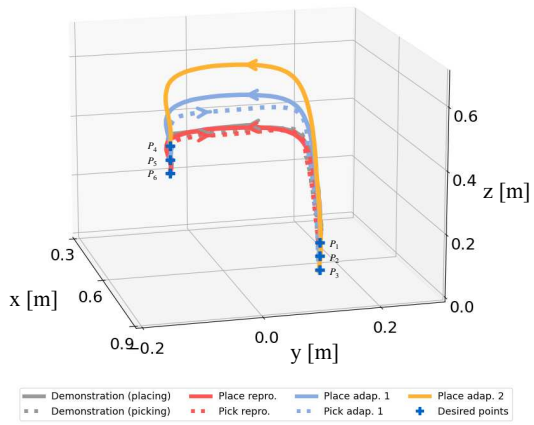

Fig. 16: The demonstrations for placing and picking tasks as well as real robotic trajectories throughout moving three blocks. The arrows depict movement directions.

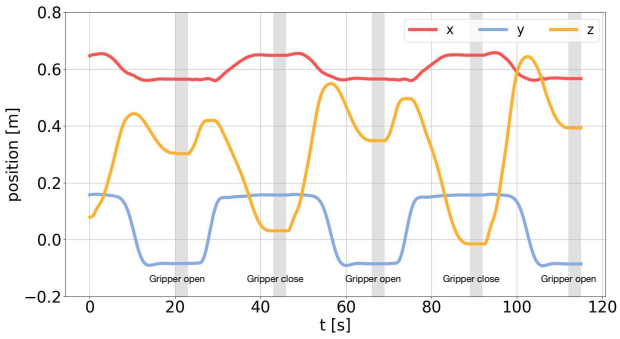

Fig. 17: The real robotic trajectories against time in the block-stacking task, where the shaded areas highlight the status of the gripper attached to the robot's end-effector.

another stack until all blocks are moved. Specifically, the task consists of five sub-tasks: placing block ③ → picking block ② → placing block ② → picking block ① → placing block ①. Note that appropriate and precise placement of each block onto the center of another stack is required to avoid falling over and mitigate accumulated errors through the multi-step task.

We first collect one demonstration for the placing (see the first row of Fig. 15) and picking tasks (see the second row of Fig. 15), respectively. The trajectories of demonstrations are plotted in Fig. 16. After that, we solve such a long-horizon task by sequencing multiple autoDMPs, where each subtask corresponds to an autoDMP with varied starting and ending points. For each autoDMP, BO performs 300 iterations. Specifically, we consider the following desired points as illustrated in Fig. 16: (*i*) place reproduction, moving the block ③ from the start-point P_1 to the end-point P_6; (*ii*) pick reproduction, reaching the block ② located at the end-point P_2; (*iii*) place adaptation, moving the block ② from the new start-point P_2 to the new end-point P_5; (*iv*) pick adaptation, reaching the block ① at the new target point P_3; (*v*) place adaptation, moving the block ① from P_3 to the new target point P_4. Note that the subtasks in (*iii*)-(*v*) are unseen in the placing and picking demonstrations.The experimental snapshots are provided in the third–seventh rows of Fig. 15 with each row representing a subtask from (*i*) until (*v*).

The real robotic trajectories for the block-stacking task are plotted in Fig. 16, where the picking movement resembles the demonstrated picking task and the placing movement resembles the demonstrated placing task – evidencing our method's generalization capability in new tasks. We also show the curves of $t-x$, $t-y$ and $t-z$ in Fig. 17 to display the gripper's status during the block-stacking task.

3) Pushing task: The pushing task involves two subtasks: reaching the small block at a desired location and pushing it towards a desired target. We can solve such a task by setting three desired points: a start-point describing the initial state of the robot's gripper, a via-point specifying the location of the block, and an end-point defining the target. In contrast to DMP, KMP provides a straightforward way to incorporate a desired via-point, so we implement KMP within our framework (i.e., autoKMP) to accomplish the pushing task.

The procedure of collecting a demonstration is illustrated in the first row of Fig. 18. We collect five demonstrations for the pushing task and subsequently use GMM and GMR to extract a probabilistic reference trajectory, depicted by the grey curve in Fig. 19(*a*). We consider two settings for adaptation evaluations and both require new start-, via-, and end-points that are away from the reference trajectory. In addition to autoKMP, we study the performance of the MAP metric as a baseline, where both autoKMP and MAP involve 200 iterations of BO. The adapted robotic trajectories are plotted in Fig. 19(*a*), where the trajectories (plotted by the yellow and green curves) optimized with the MAP metric pass through different desired points precisely, whereas the trajectory shapes have significant distortions around the desired via-points. In contrast, the trajectories (plotted by the red and blue curves) generated by autoKMP go through the desired points while keeping the shape of the reference trajectory.

The experimental snapshots, corresponding to the second evaluation scenario in Fig. 19(*a*), are given in Fig. 18. In the second row of Fig. 18, the robot equipped with autoKMP can push the block from a new desired via-point to a new desired end-point successfully. In the third row of Fig. 18, using the MAP metric the robot can first reach the block but soon lose physical contact when the robot bypasses the block (see the distortions in Fig. 19(*a*) as well), thus failing to push the block towards the target. For more experimental details of the peg-in-hole, block-stacking, and pushing tasks, please refer to the video[1].

We emphasize that the pushing segment of the second adaptation (i.e., the blue curve in Fig. 19(*b*)) lies beyond the region covered by the dataset used for training the trajectory encoder network. For the sake of clear observation, we plot some representative training samples in Fig. 19(*b*), while the remaining samples are confined within the space spanned by these samples. Thus, autoKMP shows an extrapolation capability, allowing for reliable generalization outside the region of the training dataset for the encoder.

VI. Discussion

Given a training dataset for learning the trajectory encoder network, autoLfD can generate proper trajectories straightfor-

[1]Video: https://youtu.be/ciGUtwVe-QA

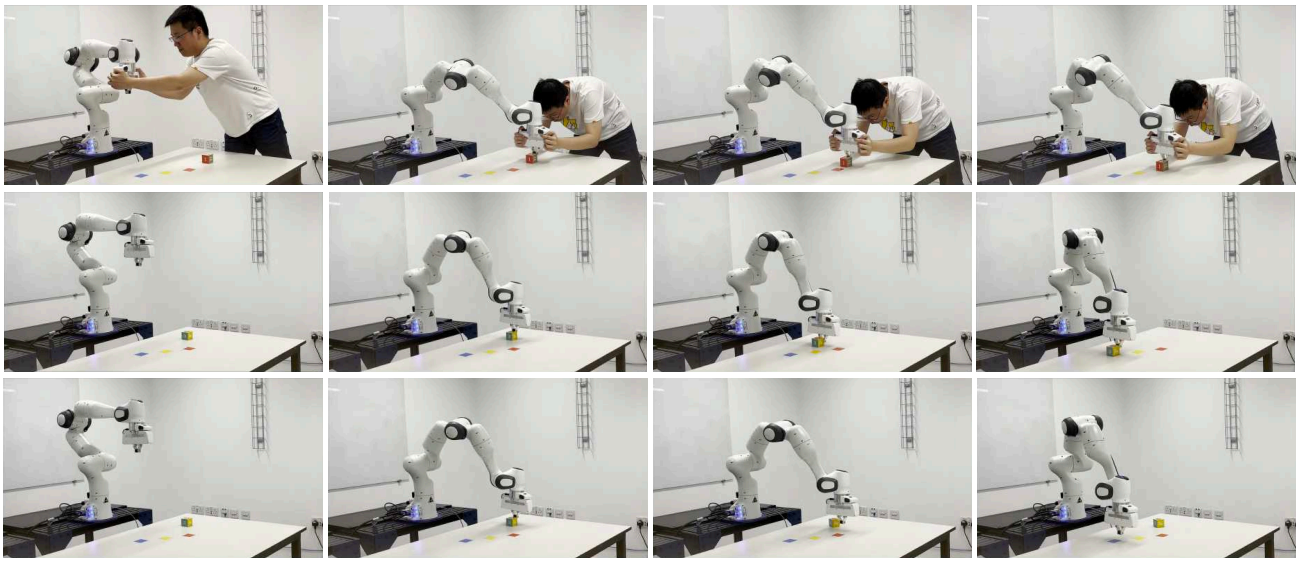

Fig. 18: Snapshots of the pushing task. *First row* illustrates the process of collecting a demonstration. *Second and third rows* show the robot executing the adapted trajectories obtained from our metric and the MAP metric, respectively.

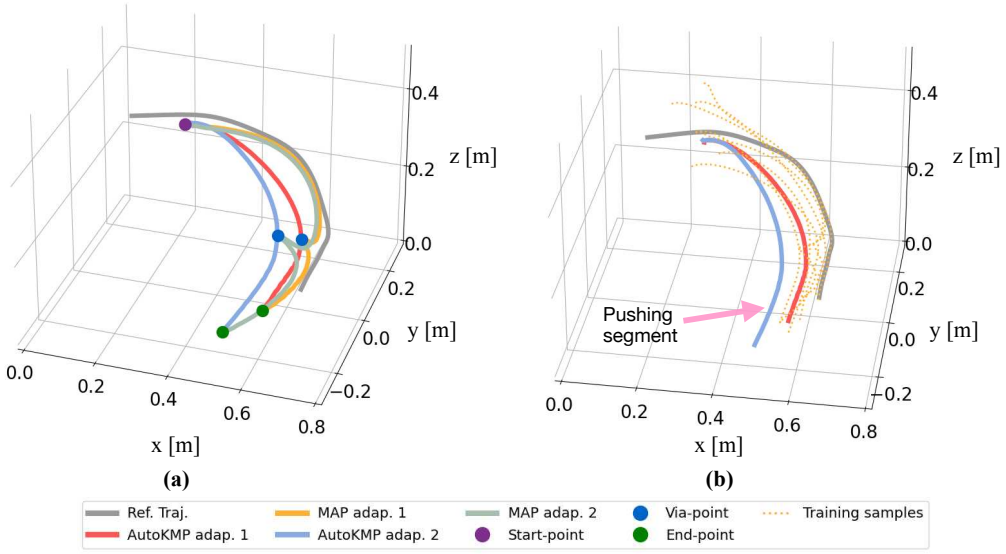

Fig. 19: The evaluations of autoKMP and the MAP metric in the pushing task. (*a*) shows two groups of evaluations and in either group both our metric (i.e., autoKMP) and the MAP metric are employed in hyperparameter optimization, respectively; (*b*) plots some representative samples for training the trajectory encoder network, while the remaining samples stay within the space formed by these samples.

wardly for unseen tasks without interacting with the environment or learning from further human guidance. In some previous works, e.g., [30]–[33], the (hyper-)parameters of an LfD method are refined using reinforcement learning, where the interaction with the environment is required and an explicit definition of reward function is also needed. Considering the 2-D writing task illustrated in Fig. 1, it is nontrivial to define a proper reward function to measure the similarity between the adapted trajectory and the demonstration, given that the typical distance-aware metrics (including MSE, DTW and Fréchet distance) are unable to provide a reliable indicator of such generalization (see Fig. 8 and Fig. 9). To take the peg-in-hole task as another example, [34] manually designed a reward function to account for errors during hole search and peg insertion and utilized RL to optimize the parameters of DMP through plenty of interactions between the peg and the hole. [35] integrated DMP with a residual policy to enhance its overall performance in the peg-in-hole task, where many training episodes were also needed. In contrast to the aforementioned works that improve DMP via learning from trial and error (i.e., the core principle of RL), autoLfD employs a different paradigm: the optimal trajectories are obtained without any interaction with the environment and can be directly and successfully deployed in real-world tasks (see the peg-in-hole task in Section V-B1).

In addition to RL, there are other attempts to optimize the hyperparameter of LfD. However, a suitable metric or objective function to indicate the performance of hyperparameter choices is always required, regardless of the optimization techniques employed (e.g., gradient descent, Bayesian optimization, and grid search) [36]–[38]. In [28], the hyperparameters of DMP were optimized using a metric that combines the root mean squared error and the end-point error. [39] optimized the hyperparameters of KMP by maximizing the reproduction probability of demonstrations. However,

these works usually ignore the shape maintenance requirement in imitation learning, although the adapted trajectories are expected to be semantically similar to the demonstrations. We also showed the limitations of various distance-aware metrics, such as MSE, DTW, and Fréchet distance [40], [41] in Section V-A. Unlike these metrics, our metric demonstrates superior performance in evaluating the generalization capability of LfD (not only limited to DMP) and can effectively guide the auto-optimization of LfD across different tasks, including writing, peg-in-hole, block-stacking, and pushing tasks – all are evidenced in Section V.

In Fig. 5, autoDMP shows reliable performance in terms of adapting the demonstration outside of the demonstrated region, which is well-known as an extrapolation problem. TP-GMM and its variants were developed to deal with the extrapolation issue [12], [26], [42]. However, TP-GMM maintains a trajectory shape within each task frame. When the final trajectory is retrieved by calculating the Gaussian product of trajectory distributions from different task frames, the shape of the ultimate trajectory may become distorted, as depicted in Fig. 8. A recent work targeting extrapolation and shape maintenance is equation learner network (EQLN) [43], where a set of activation functions were used to approximate the analytic function that underlies demonstrations, ensuring that the shape of the demonstration is maintained. In contrast to [43], we provide a generic metric to evaluate the generalization performance and the metric can be combined with various LfD approaches to inherit their merits. For example, in Fig. 6, we can generate an adapted trajectory going through a new via-point in addition to new starting and ending points, while maintaining the shape of the demonstration. The capability to pass through unseen via-points is a key feature provided by KMP. Note that the proposed closed-loop framework can be extended to many other LfD approaches in a complementary way, e.g., ProMP, TP-GMM (i.e., optimizing the parameters of task frames, see [26]), and EQLN [43]. Another work built on deep learning is conditional neural movement primitives, whereas it is unable to handle the extrapolation problem, as pointed out in [43].

The idea of projecting demonstrations into latent space has been exploited in previous works. For instance, in [44], [45], demonstrations were projected into a latent space and later a policy conditioned on latent features was learned. [46] studied continual multitask learning accounting for incomplete demonstrations, where the latent features of complete and incomplete demonstrations for the same task were enforced to stay close while the latent features of demonstrations from different tasks stayed away from each other. In autoLfD, we study the latent representation of demonstrations for the purpose of measuring the generalization performance of LfD methods. Namely, in the latent space, desired adaptations should stay close to demonstrations while undesired adaptations should stay away from demonstrations.

While we only focus on trajectory adaptation associated with time input in Euclidean space, further extensions of this work could address more settings to endow robots with more skills, including learning of force/torque demonstrations [7], constrained skill learning [47], [48], null-space learning [49], obstacle avoidance [50], human-robot collaboration [47], [51], long-horizon skills [52], and geometry-aware skills (such as orientation [4] and stiffness matrix [5]).

VII. Limitations

There are several limitations that may restrict the application of our method. To take the letter-writing tasks in Section V-A as an example, the trained encoder network can effectively measure the similarity between 2-D English letters. However, the encoder network will become inappropriate in new writing tasks that involve very different trajectories (e.g., writing Chinese characters). In general, new tasks that deviate significantly from the training data used for the trajectory encoder network correspond to the out-of-distribution problem. Addressing such tasks requires a new dataset that adequately covers the space of these new tasks.

In our experiments, we labelled samples in 2-D and 3-D spaces, which is intuitive and straightforward. However, for high-dimensional trajectories (e.g., 7 degrees of freedom joint trajectory), the labelling process could be challenging and the time cost of labelling samples will increase dramatically with the dimension of the trajectory. Moreover, it is nontrivial to determine the number of positive and negative samples for the training dataset. In practice, we collect more samples if the trajectory network does not act satisfactorily.

VIII. Conclusions

In this paper, we have introduced a closed-loop framework autoLfD allowing for optimizing the hyperparameters of LfD in an automatic manner, where a novel metric that measures the generalization performance of LfD is developed. Unlike the widely used MSE, MAP, Fréchet distance and DTW metrics, our metric acts as a reliable indicator when evaluating task adaptations. The performance of autoLfD has been verified on DMP and KMP through various tasks, including the writing, peg-in-hole, block-stacking, and pushing tasks.

References

[1] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *Proc. International Conference on Machine Learning*, 1997, pp. 12–20.

[2] M. Saveriano, F. J. Abu-Dakka, A. Kramberger, and L. Peternel, "Dynamic movement primitives in robotics: A tutorial survey," *The International Journal of Robotics Research*, vol. 42, no. 13, pp. 1133–1184, 2023.

[3] J. Zhu, M. Gienger, and J. Kober, "Learning task-parameterized skills from few demonstrations," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4063–4070, 2022.

[4] M. J. Zeestraten, I. Havoutis, J. Silvério, S. Calinon, and D. G. Caldwell, "An approach for imitation learning on riemannian manifolds," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1240–1247, 2017.

[5] F. J. Abu-Dakka, Y. Huang, J. Silvério, and V. Kyrki, "A probabilistic framework for learning geometry-based robot manipulation skills," *Robotics and Autonomous Systems*, vol. 141, p. 103761, 2021.

[6] M. Saveriano, F. J. Abu-Dakka, and V. Kyrki, "Learning stable robotic skills on riemannian manifolds," *Robotics and Autonomous Systems*, vol. 169, 2023.

[7] A. Kramberger, A. Gams, B. Nemec, D. Chrysostomou, O. Madsen, and A. Ude, "Generalization of orientation trajectories and force-torque profiles for robotic assembly," *Robotics and autonomous systems*, vol. 98, pp. 333–346, 2017.

[8] C. Yang, C. Zeng, C. Fang, W. He, and Z. Li, "A dmps-based framework for robot learning and generalization of humanlike variable impedance skills," *IEEE/ASME Transactions on Mechatronics*, vol. 23, no. 3, pp. 1193–1203, 2018.

[9] C. Yang, J. Luo, C. Liu, M. Li, and S.-L. Dai, "Haptics electromyography perception and learning enhanced intelligence for teleoperated robot," *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 4, pp. 1512–1521, 2018.

[10] H. Kim, C. Oh, I. Jang, S. Park, H. Seo, and H. J. Kim, "Learning and generalizing cooperative manipulation skills using parametric dynamic movement primitives," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 4, pp. 3968–3979, 2022.

[11] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: learning attractor models for motor behaviors," *Neural computation*, vol. 25, no. 2, pp. 328–373, 2013.

[12] S. Calinon, "A tutorial on task-parameterized movement learning and retrieval," *Intelligent Service Robotics*, vol. 9, pp. 1–29, 2016.

[13] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, "Probabilistic movement primitives," in *Proc. Advances in Neural Information Processing Systems*, 2013, pp. 2616–2624.

[14] Y. Huang, L. Rozo, J. Silvério, and D. G. Caldwell, "Kernelized movement primitives," *The International Journal of Robotics Research*, vol. 38, no. 7, pp. 833–852, 2019.

[15] H. Alt and M. Godau, "Computing the fréchet distance between two polygonal curves," *International Journal of Computational Geometry & Applications*, vol. 5, no. 01n02, pp. 75–91, 1995.

[16] K. Wang and T. Gasser, "Alignment of curves by dynamic time warping," *The annals of Statistics*, vol. 25, no. 3, pp. 1251–1276, 1997.

[17] Y. Fanger, J. Umlauft, and S. Hirche, "Gaussian processes for dynamic movement primitives with application in knowledge-based cooperation," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2016, pp. 3913–3919.

[18] C. E. Rasmussen and C. K. Williams, *Gaussian Processes for Machine Learning*. MIT press, 2006.

[19] D. A. Cohn, Z. Ghahramani, and M. I. Jordan, "Active learning with statistical models," *Journal of Artificial Intelligence Research*, vol. 4, pp. 129–145, 1996.

[20] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 815–823.

[21] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a "siamese" time delay neural network," in *Proc. Advances in Neural Information Processing Systems*, 1993.

[22] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *Proc. International Conference on Computer Vision and Pattern Recognition*, 2005, pp. 539–546.

[23] P. I. Frazier, "A tutorial on bayesian optimization," *arXiv preprint arXiv:1807.02811*, 2018.

[24] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global optimization*, vol. 13, pp. 455–492, 1998.

[25] S. Calinon and D. Lee, "Learning control," in *Humanoid robotics: A reference*. Springer, 2017.

[26] Y. Huang, J. Silvério, L. Rozo, and D. G. Caldwell, "Generalized task-parameterized skill learning," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 5667–5474.

[27] T. Kulvicius, K. Ning, M. Tamosiunaite, and F. Worgötter, "Joining movement sequences: Modified dynamic movement primitives for robotics applications exemplified on handwriting," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 145–157, 2011.

[28] L. Panchetti, J. Zheng, M. Bouri, and M. Mielle, "Team: a parameter-free algorithm to teach collaborative robots motions from user demonstrations," *arXiv preprint arXiv:2209.06940*, 2022.

[29] S. Niekum, S. Osentoski, G. Konidaris, S. Chitta, B. Marthi, and A. G. Barto, "Learning grounded finite-state representations from unstructured demonstrations," *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 131–157, 2015.

[30] J. Kober, E. Oztop, and J. Peters, "Reinforcement learning to adjust robot movements to new situations," *Robotics: Science and Systems, MIT Press Journal*, vol. 6, pp. 33–40, 2011.

[31] F. Guenter, M. Hersch, S. Calinon, and A. Billard, "Reinforcement learning for imitating constrained reaching movements," *Advanced Robotics*, vol. 21, no. 13, pp. 1521–1544, 2007.

[32] F. Stulp and O. Sigaud, "Robot skill learning: From reinforcement learning to evolution strategies," *Paladyn, Journal of Behavioral Robotics*, vol. 4, no. 1, pp. 49–61, 2013.

[33] X. Sun, J. Li, A. V. Kovalenko, W. Feng, and Y. Ou, "Integrating reinforcement learning and learning from demonstrations to learn nonprehensile manipulation," *IEEE Transactions on Automation Science and Engineering*, 2022.

[34] N. J. Cho, S. H. Lee, J. B. Kim, and I. H. Suh, "Learning, improving, and generalizing motor skills for the peg-in-hole tasks based on imitation learning and self-learning," *Applied Sciences*, vol. 10, no. 8, p. 2719, 2020.

[35] T. Davchev, K. S. Luck, M. Burke, F. Meier, S. Schaal, and S. Ramamoorthy, "Residual learning from demonstration: Adapting dmps for contact-rich manipulation," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4488–4495, 2022.

[36] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, pp. 295–316, 2020 .

[37] M. Moll, C. Chamzas, Z. Kingston, and L. E. Kavraki, "Hyperplan: A framework for motion planning algorithm selection and parameter optimization," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 2511–2518.

[38] L. Hussenot, M. Andrychowicz, D. Vincent, R. Dadashi, A. Raichuk, S. Ramos, N. Momchev, S. Girgin, R. Marinier, L. Stafiniak *et al.*, "Hyperparameter selection for imitation learning," in *International Conference on Machine Learning*. PMLR, 2021, pp. 4511–4522.

[39] A. Liu, S. Zhan, Z. Jin, and W.-a. Zhang, "A variable impedance skill learning algorithm based on kernelized movement primitives," *IEEE Transactions on Industrial Electronics*, 2023.

[40] Z. Zhang, K. Huang, and T. Tan, "Comparison of similarity measures for trajectory clustering in outdoor surveillance scenes," in *18th International Conference on Pattern Recognition (ICPR'06)*, vol. 3. IEEE, 2006, pp. 1135–1138.

[41] B. Hertel and S. R. Ahmadzadeh, "Similarity-aware skill reproduction based on multi-representational learning from demonstration," in *Proc. International Conference on Advanced Robotics*, 2021, pp. 652–657 .

[42] S. Calinon, D. Bruno, and D. G. Caldwell, "A task-parameterized probabilistic model with minimal intervention control," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 3339–3344.

[43] H. Perez-Villeda, J. Piater, and M. Saveriano, "Learning and extrapolation of robotic skills using task-parameterized equation learner networks," *Robotics and Autonomous Systems*, vol. 160, p. 104309, 2023.

[44] C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, and P. Sermanet, "Learning latent plans from play," in *Conference on robot learning*. PMLR, 2020, pp. 1113–1132.

[45] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn, "Bc-z: Zero-shot task generalization with robotic imitation learning," in *Conference on Robot Learning*. PMLR, 2022, pp. 991–1002.

[46] M. B. Hafez and S. Wermter, "Continual robot learning using self-supervised task inference," *IEEE Transactions on Cognitive and Developmental Systems*, 2023.

[47] R. Huang, H. Cheng, J. Qiu, and J. Zhang, "Learning physical human–robot interaction with coupled cooperative primitives for a lower exoskeleton," *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 4, pp. 1566–1574, 2019.

[48] M. Saveriano and D. Lee, "Learning barrier functions for constrained motion planning with dynamical systems," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 112–119.

[49] J. Silvério and Y. Huang, "A non-parametric skill representation with soft null space projectors for fast generalization," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 2988–2994.

[50] S. Xiao, X. Chen, Y. Lu, J. Ye, and H. Wu, "A kmp-based interactive learning approach for robot trajectory adaptation with obstacle avoidance," *Industrial Robot: the international journal of robotics research and application*, 2024.

[51] Y. Cui, J. Poon, J. V. Miro, K. Yamazaki, K. Sugimoto, and T. Matsubara, "Environment-adaptive interaction primitives through visual context for human–robot motor skill learning," *Autonomous Robots*, vol. 43, pp. 1225–1240, 2019.

[52] H. Wu, W. Yan, Z. Xu, T. Cheng, and X. Zhou, "A framework of robot skill learning from complex and long-horizon tasks," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 4, pp. 3628–3638, 2021.