A Computationally Efficient Nonparametric Approach for Robot Imitation Learning

Yijin Wang[1], Shaokang Wu[1], Chen Liu[1], Chuankai Zhang[1], João Silvério[2], and Yanlong Huang[1*]

***Abstract*— Transferring human skills to robots through learning from demonstrations has been an important topic in the robotics community, and many models have been developed for learning and adapting such skills. Among them, nonparametric representations are an appealing choice, since nonparametric solutions alleviate the explicit definition of basis functions, require fewer hyperparameters, and facilitate straightforward generalization for tasks involving high-dimensional inputs (e.g., human–robot collaboration and dual-arm manipulation). However, a commonly raised concern for nonparametric models is their computational complexity. In this paper, we propose a computationally efficient solution for nonparametric skill learning, whose computation time grows quadratically with the length of demonstrations, as opposed to the cubic growth in a standard nonparametric model. The solution is further improved by exploiting local models and fusing their predictions. We evaluate our approach in a 2-D writing task with time input, a 3-D human–guided obstacle avoidance task, and a dual-arm transportation task associated with 7-D input. The results show that our solution achieves comparable performance to the parametric method and enables instant adaptations in tasks associated with time or multi-dimensional inputs.**

I. Introduction

Numerous works from the community of robot learning have aimed to endow robots with strong generalization capability. In particular, learning from human demonstrations has been exploited in various aspects, e.g., learning of Cartesian positions [1], manifold learning for orientation [2], constrained learning with hard [3] and soft constraints [4], and stable dynamical systems [5].

To transfer human demonstrations to robots effectively, a proper model encoding human skills is crucial, as the model will be subsequently employed to deal with unseen situations. Under different assumptions, a variety of algorithms have been proposed. For example, dynamical movement primitives (DMP) [1] use a second-order dynamical system to learn the mapping from position and velocity to acceleration. Task-parameterized Gaussian mixture model [6] (TP-GMM) leverages GMM to learn the relative motion pattern in each task frame. Probabilistic movement primitives (ProMP) [7] model demonstrations as a weighted sum of multiple basis functions. TP-GMM has proven effective in extrapolation situations; it may become inappropriate for tasks that require adaptations passing through desired regions. While DMP and ProMP offer flexible adaptation features, it is nontrivial to extend them to deal with demonstrations associated with multi-dimensional inputs, e.g., in bimanual tasks where one robotic arm must react appropriately to the other arm's actions, or in human-robot collaboration tasks where the robot must respond promptly to human actions. As both methods depend on fixed basis functions, the required number of such functions often increases quickly as the size of the inputs grows [8]. Differing from the parametric models, nonparametric methods, such as kernelized movement primitives (KMP) [9] and the local heteroscedastic Gaussian process (HGP) [10], provide an alternative way to learn demonstrations.

A key issue arising from nonparametric models is computational complexity. In a standard GP, it requires time $\mathcal{O}(N^3)$ with N being the size of the demonstrations. In KMP, the time complexity is higher as it encodes the correlations among outputs and predicts both the mean and full covariance matrix for a query input, therefore it requires $\mathcal{O}(\mathcal{D}^3N^3)$, where $\mathcal{D}$ denotes the dimension of outputs. The time incurred by GP and KMP is mainly due to inverting the kernel matrix. While matrix inverse can be precomputed for a fixed training dataset or demonstrations, it must be recomputed whenever the dataset or demonstrations are updated by adding new datapoints or removing old ones. To address this issue, some methods were developed to speed up GP computation, such as sparse GP approximation [11], sliding window incremental update [12], and local GP [13] with Cholesky incremental update [14].

Recently, a fast version of KMP was proposed in [15], which takes advantage of the properties of the null-space projector and reduces the computational complexity to $\mathcal{O}(\mathcal{D}^3N^2)$. However, the null-space KMP (NS-KMP) has limited adaptation accuracy, as it formulates adaptation requirements as secondary objectives and therefore adapts demonstrations only towards the desired region without generating trajectories that accurately pass the desired region, as achieved by standard KMP. While KMP offers more adaptation features than GP for learning demonstrations [3], [9], [16], [17], e.g., capturing the probabilistic characteristics of multiple demonstrations, the correlation among multiple outputs, and both epistemic and aleatoric uncertainties [18], as well as enabling accurate adaptations to arbitrary regions, we focus on developing a computationally efficient imitation learning solution built on KMP.

Inspired by the kernel-based recursive least-squares

[1]School of Computer Science, University of Leeds, Leeds LS2 9JT, UK. *Corresponding author: y.l.huang@leeds.ac.uk

[2]German Aerospace Center (DLR), Robotics and Mechatronics Center (RMC), Munchner Str. 20, 82234 Weßling, Germany. joao.silverio@dlr.de

This work was supported by the Royal Society under Grant RG/R1/251228 and by the European Union's Horizon Research and Innovation Programme under Grant 101136067 (INVERSE).

(KRLS) algorithm [12], [19], we propose an *instant KMP* (iKMP) which allows for fast online adaptation without inverting the full kernel matrix (Section III). iKMP is identical to standard KMP but requires significantly less computation time. Furthermore, we follow the spirit of local GP [13] and develop a *mixture of instant KMPs* (MiKMP) to further improve the computational efficiency (Section IV). The computational complexity analysis of iKMP and MiKMP (Section V) shows that they require significantly less time than standard KMP. Finally, we validate our methods (including comparison with two additional baselines) through a 2-D writing task, a 3-D obstacle avoidance task, and a dual-arm transportation task involving demonstrations with 7-D inputs (Section VI).

II. Kernelized Movement Primitives

A. KMP

Supposing that we have collected H demonstrations $\{\{\mathbf{s}_{n,h}, \boldsymbol{\xi}_{n,h}\}_{n=1}^{N}\}_{h=1}^{H}$, where N denotes the length of a demonstration, $\mathbf{s}_{n,h} \in \mathbb{R}^{\mathcal{I}}$ represents the input (e.g., time in a time-driven task or human hand pose in a human-robot collaboration task), and $\boldsymbol{\xi}_{n,h} \in \mathbb{R}^{\mathcal{D}}$ denotes the output (e.g., robot end-effector's pose). KMP first employs GMM to capture the joint probability distribution $\mathcal{P}(\mathbf{s}, \boldsymbol{\xi})$ from demonstrations, and later uses Gaussian mixture regression (GMR) to estimate the conditional distribution $\mathcal{P}(\boldsymbol{\xi}|\mathbf{s})$. Specifically, a *probabilistic reference trajectory* with N datapoints $\{\hat{\boldsymbol{\mu}}_n, \hat{\boldsymbol{\Sigma}}_n\}_{n=1}^{N}$ can be retrieved via GMR, where $\boldsymbol{\xi}_n|\mathbf{s}_n \sim \mathcal{N}(\hat{\boldsymbol{\mu}}_n, \hat{\boldsymbol{\Sigma}}_n)$. The reference trajectory $\mathbf{L} = \{\mathbf{s}_n, \hat{\boldsymbol{\mu}}_n, \hat{\boldsymbol{\Sigma}}_n\}_{n=1}^{N}$ encapsulates the distribution of demonstrations, and KMP learns such a distribution instead of original demonstrations.

For a query input $\mathbf{s}^*$, KMP predicts the mean and covariance for its corresponding output $\boldsymbol{\xi}(\mathbf{s}^*)$ as [9]

$$\begin{aligned} \mathbb{E}(\boldsymbol{\xi}(\mathbf{s}^*)) &= \mathbf{k}^*(\mathbf{K} + \lambda_m\boldsymbol{\Sigma})^{-1}\boldsymbol{\mu}, \\ \mathbb{D}(\boldsymbol{\xi}(\mathbf{s}^*)) &= \alpha\left(\mathbf{k}(\mathbf{s}^*, \mathbf{s}^*) - \mathbf{k}^*(\mathbf{K} + \lambda_c\boldsymbol{\Sigma})^{-1}\mathbf{k}^{*\top}\right), \end{aligned} \tag{1}$$

where $\lambda_m > 0, \lambda_c > 0$, and $\alpha > 0$ are hyperparameters. $\boldsymbol{\mu} = [\hat{\boldsymbol{\mu}}_1^\top \ \hat{\boldsymbol{\mu}}_2^\top \ \cdots \ \hat{\boldsymbol{\mu}}_N^\top]^\top$ and $\boldsymbol{\Sigma} = \text{blockdiag}(\hat{\boldsymbol{\Sigma}}_1, \hat{\boldsymbol{\Sigma}}_2, \ldots, \hat{\boldsymbol{\Sigma}}_N)$. The matrices $\mathbf{k}^*$ and $\mathbf{K}$ are separately calculated by

$$\mathbf{k}^* = [\mathbf{k}(\mathbf{s}^*, \mathbf{s}_1)\ \mathbf{k}(\mathbf{s}^*, \mathbf{s}_2)\ \cdots\ \mathbf{k}(\mathbf{s}^*, \mathbf{s}_N)], \tag{2}$$

$$\mathbf{K} = \begin{bmatrix} \mathbf{k}(\mathbf{s}_1, \mathbf{s}_1) & \mathbf{k}(\mathbf{s}_1, \mathbf{s}_2) & \cdots & \mathbf{k}(\mathbf{s}_1, \mathbf{s}_N) \\ \mathbf{k}(\mathbf{s}_2, \mathbf{s}_1) & \mathbf{k}(\mathbf{s}_2, \mathbf{s}_2) & \cdots & \mathbf{k}(\mathbf{s}_2, \mathbf{s}_N) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{k}(\mathbf{s}_N, \mathbf{s}_1) & \mathbf{k}(\mathbf{s}_N, \mathbf{s}_2) & \cdots & \mathbf{k}(\mathbf{s}_N, \mathbf{s}_N) \end{bmatrix}, \tag{3}$$

where

$$\mathbf{k}(\mathbf{s}_i,\ \mathbf{s}_j) = k(\mathbf{s}_i, \mathbf{s}_j)\mathbf{I}_{\mathcal{D}}, \tag{4}$$

with $k(\cdot, \cdot)$ being a kernel function.

B. Skill Generalization using KMP

Since only a limited number of demonstrations are available, the generalization capability of imitation learning is highly desirable, which permits a robot to adapt the learned skills from demonstrations to unseen tasks. Assuming that a new task imposes adaptation requirements in the form of M desired points, i.e., $\boldsymbol{\xi}(\bar{\mathbf{s}}_m)|\bar{\mathbf{s}}_m \sim \mathcal{N}(\bar{\boldsymbol{\mu}}_m, \bar{\boldsymbol{\Sigma}}_m), m \in \{1, 2, \ldots, M\}$, KMP addresses both 'imitation learning' and the 'adaptation requirements' by learning an *updated refererence trajectory*. For each adaptation requirement $(\bar{\mathbf{s}}_m, \bar{\boldsymbol{\mu}}_m, \bar{\boldsymbol{\Sigma}}_m)$, the reference trajectory is updated by

$$\begin{cases} \mathbf{L} \leftarrow \{\mathbf{L}/\{\mathbf{s}_r, \hat{\boldsymbol{\mu}}_r, \hat{\boldsymbol{\Sigma}}_r\}\} \cup \{\bar{\mathbf{s}}_m, \bar{\boldsymbol{\mu}}_m, \bar{\boldsymbol{\Sigma}}_m\}, & \text{if } d(\bar{\mathbf{s}}_m, \mathbf{s}_r) < \zeta, \\ \mathbf{L} \leftarrow \mathbf{L} \cup \{\bar{\mathbf{s}}_m, \bar{\boldsymbol{\mu}}_m, \bar{\boldsymbol{\Sigma}}_m\}, & \text{otherwise}, \end{cases} \tag{5}$$

where $r = \text{argmin}_n\, d(\bar{\mathbf{s}}_m, \mathbf{s}_n), n \in \{1, 2, \ldots, N\}$ with $\zeta > 0$ representing a distance threshold and $d(\cdot)$ being a distance function, such as ℓ_2 norm. The operation '/' denotes removing an element from a set, while '$\cup$' denotes adding an element to a set.

III. Instant KMP

Learning an updated reference trajectory to address the adaptation requirements via KMP results in new matrices $\mathbf{k}^*$, $\mathbf{K}$, $\boldsymbol{\Sigma}$, and $\boldsymbol{\mu}$. In particular, the most computationally expensive parts in (1) are $(\mathbf{K}+\lambda_m\boldsymbol{\Sigma})^{-1}$ and $(\mathbf{K}+\lambda_c\boldsymbol{\Sigma})^{-1}$. In the following discussion, we focus on efficient computation for $(\mathbf{K} + \lambda_m\boldsymbol{\Sigma})^{-1}$, while $(\mathbf{K} + \lambda_c\boldsymbol{\Sigma})^{-1}$ can be calculated similarly. Our solution builds on the KRLS algorithm in [12], [19], which studied fast learning of datapoints within a sliding window via efficient matrix-inversion methods. We leverage these mechanisms to enable instant adaptations for KMP.

For simplicity, we take the first desired point $(\bar{\mathbf{s}}_1, \bar{\boldsymbol{\mu}}_1, \bar{\boldsymbol{\Sigma}}_1)$ as an example. We assume that the corresponding update of the reference trajectory involves both 'removing' and 'adding' operations, and the nearest datapoint in $\mathbf{L}$ to $(\bar{\mathbf{s}}_1, \bar{\boldsymbol{\mu}}_1, \bar{\boldsymbol{\Sigma}}_1)$ is $(\mathbf{s}_r, \hat{\boldsymbol{\mu}}_r, \hat{\boldsymbol{\Sigma}}_r)$. We consider removing the nearest datapoint from the reference trajectory in Section III-A and adding the desired point to it in Section III-B.

A. Excluding a Datapoint from the Reference Trajectory

Expanding the term $\mathbf{K} + \lambda_m\boldsymbol{\Sigma}$, we have

$$\mathbf{K} + \lambda_m\boldsymbol{\Sigma} = \begin{bmatrix} \mathbf{k}(\mathbf{s}_1, \mathbf{s}_1) + \lambda_m\hat{\boldsymbol{\Sigma}}_1 & \cdots & \mathbf{k}(\mathbf{s}_1, \mathbf{s}_r) & \cdots & \mathbf{k}(\mathbf{s}_1, \mathbf{s}_N) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{k}(\mathbf{s}_r, \mathbf{s}_1) & \cdots & \mathbf{k}(\mathbf{s}_r, \mathbf{s}_r) + \lambda_m\hat{\boldsymbol{\Sigma}}_r & \cdots & \mathbf{k}(\mathbf{s}_r, \mathbf{s}_N) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{k}(\mathbf{s}_N, \mathbf{s}_1) & \cdots & \mathbf{k}(\mathbf{s}_N, \mathbf{s}_r) & \cdots & \mathbf{k}(\mathbf{s}_N, \mathbf{s}_N) + \lambda_m\hat{\boldsymbol{\Sigma}}_N \end{bmatrix} \tag{6}$$

After removing the nearest datapoint from $\mathbf{L}$, we encounter a new term, i.e.,

$$\mathbf{A} = \bar{\mathbf{K}} + \lambda_m\bar{\boldsymbol{\Sigma}}, \tag{7}$$

with

$$\begin{aligned} \bar{\mathbf{K}} &= \begin{bmatrix} \mathbf{k}(\mathbf{s}_1, \mathbf{s}_1) & \cdots & \mathbf{k}(\mathbf{s}_1, \mathbf{s}_{r-1}) & \mathbf{k}(\mathbf{s}_1, \mathbf{s}_{r+1}) & \cdots & \mathbf{k}(\mathbf{s}_1, \mathbf{s}_N) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{k}(\mathbf{s}_N, \mathbf{s}_1) & \cdots & \mathbf{k}(\mathbf{s}_N, \mathbf{s}_{r-1}) & \mathbf{k}(\mathbf{s}_N, \mathbf{s}_{r+1}) & \cdots & \mathbf{k}(\mathbf{s}_N, \mathbf{s}_N) \end{bmatrix}, \\ \bar{\boldsymbol{\Sigma}} &= \text{blockdiag}(\hat{\boldsymbol{\Sigma}}_1, \ldots, \hat{\boldsymbol{\Sigma}}_{r-1}, \hat{\boldsymbol{\Sigma}}_{r+1}, \ldots, \hat{\boldsymbol{\Sigma}}_N). \end{aligned} \tag{8}$$

To compute $\mathbf{A}^{-1}$ efficiently, we resort to the properties of permutation matrices and block matrix inversion. Since $\mathbf{K} + \lambda_m\boldsymbol{\Sigma}$ depends on the original reference trajectory, we can precompute its inverse $(\mathbf{K} + \lambda_m\boldsymbol{\Sigma})^{-1}$ in an *offline*

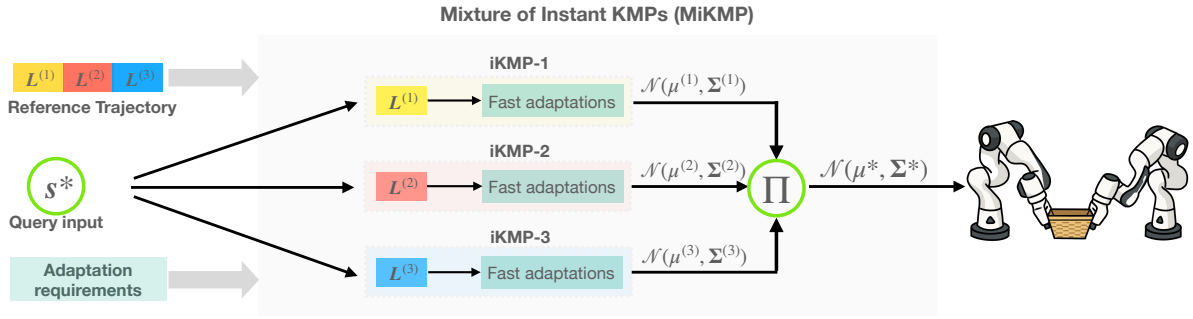

Fig. 1. Overview of MiKMP, illustrated with three iKMPs. Note that all iKMPs share the same query inputs and adaptation requirements.

manner before processing the updated reference trajectory, allowing it to be reused to facilitate the calculation of $\mathbf{A}^{-1}$.

By applying a proper permutation matrix to $\mathbf{K}+\lambda_m\mathbf{\Sigma}$, we can move its r-th block row and r-th block column to the last block row and last block column, respectively. We denote such a permulation matrix as $\mathbf{P}$; therefore,

$$\mathbf{P}(\mathbf{K}+\lambda_m\mathbf{\Sigma})\mathbf{P}^T = \left[\begin{array}{c|c} \mathbf{A} & \mathbf{B} \\ \hline \mathbf{C} & \mathbf{D} \end{array}\right], \tag{9}$$

where $\mathbf{A}$ is defined in (7),

$$\begin{aligned} &\mathbf{C} = \begin{bmatrix} \mathbf{k}(\mathbf{s}_r,\mathbf{s}_1) & \cdots & \mathbf{k}(\mathbf{s}_r,\mathbf{s}_{r-1}) & \mathbf{k}(\mathbf{s}_r,\mathbf{s}_{r+1}) & \cdots & \mathbf{k}(\mathbf{s}_r,\mathbf{s}_N) \end{bmatrix}, \\ &\mathbf{B} = \mathbf{C}^T, \\ &\mathbf{D} = \mathbf{k}(\mathbf{s}_r,\mathbf{s}_r) + \lambda_m\hat{\mathbf{\Sigma}}_r. \end{aligned} \tag{10}$$

Assuming $(\mathbf{K}+\lambda_m\mathbf{\Sigma})^{-1}$ is already known, we can compute the inverse of the matrix in (9), i.e.,

$$\left(\mathbf{P}(\mathbf{K}+\lambda_m\mathbf{\Sigma})\mathbf{P}^T\right)^{-1} = \mathbf{P}(\mathbf{K}+\lambda_m\mathbf{\Sigma})^{-1}\mathbf{P}^T = \left[\begin{array}{c|c} \widetilde{\mathbf{A}} & \widetilde{\mathbf{B}} \\ \hline \widetilde{\mathbf{C}} & \widetilde{\mathbf{D}} \end{array}\right], \tag{11}$$

where $\widetilde{\mathbf{A}},\widetilde{\mathbf{B}},\widetilde{\mathbf{C}},\widetilde{\mathbf{D}}$ denote the corresponding block matrices. Note that for the permutation matrix $\mathbf{P}$, it satisfies $\mathbf{P}\mathbf{P}^T=\mathbf{I}$.

In fact, given $(\mathbf{K}+\lambda_m\mathbf{\Sigma})^{-1}$, we can directly determine the four block matrices in (11) using the properties of permutation matrices, without matrix multiplication. To consider a toy example: $\mathbf{W} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$, $\mathbf{W}^{-1} = \begin{bmatrix} \bar{a} & \bar{b} & \bar{c} \\ \bar{d} & \bar{e} & \bar{f} \\ \bar{g} & \bar{h} & \bar{i} \end{bmatrix}$ and $\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$ (i.e., $\mathbf{P}\mathbf{W}\mathbf{P}^T$ moves the first row and first column of $\mathbf{W}$ to the end) . By rearranging the elements in $\mathbf{W}^{-1}$, we have $(\mathbf{P}\mathbf{W}\mathbf{P}^T)^{-1} = \left[\begin{array}{cc|c} \bar{e} & \bar{f} & \bar{d} \\ \bar{h} & \bar{i} & \bar{g} \\ \hline \bar{b} & \bar{c} & \bar{a} \end{array}\right]$.

Proposition 1. *Assuming $(\mathbf{K}+\lambda_m\mathbf{\Sigma})^{-1}$ is calculated beforehand for KMP and the four block matrices $\widetilde{\mathbf{A}},\widetilde{\mathbf{B}},\widetilde{\mathbf{C}},\widetilde{\mathbf{D}}$ in (11) are determined, we can efficiently compute the inverse of $\mathbf{A}$ in (10), i.e.,*

$$\mathbf{A}^{-1} = (\bar{\mathbf{K}}+\lambda_m\bar{\mathbf{\Sigma}})^{-1} = \widetilde{\mathbf{A}} - \widetilde{\mathbf{B}}\widetilde{\mathbf{D}}^{-1}\widetilde{\mathbf{C}}. \tag{12}$$

Proof: This is a direct application of the block matrix inversion [20], i.e.,

$$\begin{aligned} \left[\begin{array}{c|c} \mathbf{A} & * \\ \hline * & * \end{array}\right] &= \left(\left(\mathbf{P}(\mathbf{K}+\lambda_m\mathbf{\Sigma})\mathbf{P}^T\right)^{-1}\right)^{-1} \\ &= \left[\begin{array}{c|c} \widetilde{\mathbf{A}} & \widetilde{\mathbf{B}} \\ \hline \widetilde{\mathbf{C}} & \widetilde{\mathbf{D}} \end{array}\right]^{-1} = \left[\begin{array}{c|c} (\widetilde{\mathbf{A}}-\widetilde{\mathbf{B}}\widetilde{\mathbf{D}}^{-1}\widetilde{\mathbf{C}})^{-1} & * \\ \hline * & * \end{array}\right]. \end{aligned}$$

■

The computational cost of (12) mainly depends on matrix multiplication, as $\widetilde{\mathbf{D}}$ has the size $\mathcal{D}\times\mathcal{D}$ and its inversion takes negligible time. In contrast, the naive computation of $\mathbf{A}^{-1}$ requires inverting a matrix of size $\mathcal{D}(N-1)\times\mathcal{D}(N-1)$.

B. Adding a Datapoint to the Reference Trajectory

After excluding the nearest datapoint from $\mathbf{L}$, the updated reference trajectory becomes $\{\mathbf{s}_n,\hat{\boldsymbol{\mu}}_n,\hat{\mathbf{\Sigma}}_n\}_{n=1,n\neq r}^{N}$ and has the length $N-1$. Now, let us add the first desired point to the updated reference trajectory. In this case, inverting an extended matrix that incorporates new rows and columns incurred by the desired point is required. Specifically, we need to calculate

$$\mathbf{R} = \left[\begin{array}{c|c} \mathbf{A} & \mathbf{E} \\ \hline \mathbf{F} & \mathbf{G} \end{array}\right]^{-1}, \tag{13}$$

where

$$\begin{aligned} &\mathbf{F} = \begin{bmatrix} \mathbf{k}(\bar{\mathbf{s}}_1,\mathbf{s}_1) & \cdots & \mathbf{k}(\bar{\mathbf{s}}_1,\mathbf{s}_{r-1}) & \mathbf{k}(\bar{\mathbf{s}}_1,\mathbf{s}_{r+1}) & \cdots & \mathbf{k}(\bar{\mathbf{s}}_1,\mathbf{s}_N) \end{bmatrix}, \\ &\mathbf{E} = \mathbf{F}^T, \\ &\mathbf{G} = \mathbf{k}(\bar{\mathbf{s}}_1,\bar{\mathbf{s}}_1) + \lambda_m\bar{\mathbf{\Sigma}}_1. \end{aligned} \tag{14}$$

Here, we append the desired point to the end of the updated reference trajectory, rather than inserting it at the location of the nearest datapoint $(\mathbf{s}_r,\hat{\boldsymbol{\mu}}_r,\hat{\mathbf{\Sigma}}_r)$ that has been excluded. It can be shown that the performance of KMP remains unchanged regardless of the order of the datapoints in the reference trajectory.

Instead of naively calculating $\mathbf{R}$ in (13), we apply the property of block matrix inversion again [20]. Assuming the *Schur complement* $\mathbf{\Gamma} = \mathbf{G}-\mathbf{F}\mathbf{A}^{-1}\mathbf{E}$ is invertible, we have

$$\mathbf{R} = \begin{bmatrix} \mathbf{A}^{-1}+\mathbf{A}^{-1}\mathbf{E}\mathbf{\Gamma}^{-1}\mathbf{F}\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{E}\mathbf{\Gamma}^{-1} \\ -\mathbf{\Gamma}^{-1}\mathbf{F}\mathbf{A}^{-1} & \mathbf{\Gamma}^{-1}. \end{bmatrix} \tag{15}$$

By observing (15), we can see that its four blocks depend on the terms $\mathbf{\Gamma}^{-1},\mathbf{F}\mathbf{A}^{-1},\mathbf{A}^{-1}\mathbf{E}$ – all of them have significantly smaller computational complexity since $\mathbf{A}^{-1}$ is

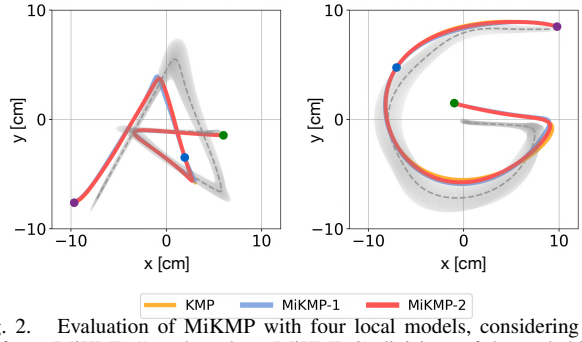

Fig. 2. Evaluation of MiKMP with four local models, considering both uniform (MiKMP-1) and random (MiKMP-2) divisions of the probabilistic reference trajectory. The dashed grey curve and the shaded area represent the mean and standard deviation of the reference trajectory. Solid dots indicate the desired points.

already obtained from *Proposition 1*. In contrast, the naive computation of $\mathbf{R}$ in (13) in general leads to a complexity of $\mathcal{O}(\mathcal{D}^3N^3)$. A detailed analysis of computation complexity will be discussed in Section V.

For the second desired point, we can follow the same procedure to cope with both 'removing' and 'adding' operations. The difference is that, instead of the original reference trajectory, we start with the updated reference trajectory consisting of $\{\mathbf{s}_n, \hat{\boldsymbol{\mu}}_n, \hat{\boldsymbol{\Sigma}}_n\}_{n=1,n\neq r}^{N}$ and $(\bar{\mathbf{s}}_1, \bar{\boldsymbol{\mu}}_1, \bar{\boldsymbol{\Sigma}}_1)$. Also, in (11), we directly use $\mathbf{R}$ instead of $(\mathbf{K}+\lambda_m\boldsymbol{\Sigma})^{-1}$. Similarly, we can process the remaining desired points separately. For a desired point that only involves the 'adding' operation, we can skip the procedure in Section III-A and only need to follow the calculations in Section III-B. Note that when implementing KMP prediction in (1), $\mathbf{k}^*$ and $\boldsymbol{\mu}$ in (1) also depend on the updated reference trajectory, but the computation cost of updating them is negligible.

IV. Mixture of Instant-KMPs

Similar to the local Gaussian process (GP) [21], in which different GPs model different regions of the training dataset, we propose learning a mixture of instant KMPs from the probabilistic reference trajectory, which further reduces the computational cost of iKMP for adaptation. An overview of MiKMP is illustrated in Fig.1.

Given the original reference trajectory $\mathbf{L}$, we can divide it into M non-overlapping segments in two ways: (i) *uniform division*, where $\mathbf{L}$ is split into consecutive segments of equal length[1]; or (ii) *random division*, where $\mathbf{L}$ is partitioned randomly into M segments. We have empirically verified both ways. In Fig. 2, the left and right plots show a mixture of four iKMPs, where MiKMP-1 corresponds to the first division method while MiKMP-2 corresponds to the second method. It can be seen that their differences are negligible. Therefore, we take the first type of division to explain the mixture of iKMPs.

Given M segments (i.e., $\mathbf{L}_1 = \{\mathbf{s}_n, \hat{\boldsymbol{\mu}}_n, \hat{\boldsymbol{\Sigma}}_n\}_{n=1}^{M}$, $\mathbf{L}_2 = \{\mathbf{s}_n, \hat{\boldsymbol{\mu}}_n, \hat{\boldsymbol{\Sigma}}_n\}_{n=M+1}^{2M}$, and so on), we can learn each segment separately using an iKMP. For a query input $\mathbf{s}^*$, the first iKMP predicts the mean and covariance for its corresponding output as $\mathbb{E}(\boldsymbol{\xi}(\mathbf{s}^*))^{(1)}$ and $\mathbb{D}(\boldsymbol{\xi}(\mathbf{s}^*))^{(1)}$. Similarly, we have the other $M-1$ groups of predictions. Since each prediction leads to a Gaussian distribution, it is a natural solution to adopt the Gaussian product to unify the predictions from different iKMPs.

Let us write $\boldsymbol{\mu}^{(m)}=\mathbb{E}(\boldsymbol{\xi}(\mathbf{s}^*))^{(m)}$ and $\boldsymbol{\Sigma}^{(m)}=\mathbb{D}(\boldsymbol{\xi}(\mathbf{s}^*))^{(m)}$. The ultimate prediction for the query point $\mathbf{s}^*$ is

$$\begin{aligned}\boldsymbol{\xi}(\mathbf{s}^*) &\sim \mathcal{N}(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*),\\ \mathcal{N}(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*) &\propto \prod_{m=1}^{M} \mathcal{N}(\boldsymbol{\mu}^{(m)}, \boldsymbol{\Sigma}^{(m)}),\end{aligned} \tag{16}$$

where

$$\boldsymbol{\Sigma}^* = (\sum_{m=1}^{M} {\boldsymbol{\Sigma}^{(m)}}^{-1})^{-1},\ \boldsymbol{\mu}^* = \boldsymbol{\Sigma}^* \sum_{m=1}^{M} {\boldsymbol{\Sigma}^{(m)}}^{-1}\boldsymbol{\mu}^{(m)}. \tag{17}$$

In (17), the mean prediction $\boldsymbol{\mu}^*$ can be interpreted as a covariance-weighted average of $\{\boldsymbol{\mu}^{(m)}\}_{m=1}^{M}$. As discussed in [18], the predicted covariance by KMP captures both variability and uncertainty in a unified manner. When the query point lies within the region that KMP has learned, the predicted covariance represents the variation of the corresponding output. The larger the predicted covariance, the less the predicted mean (after weighting by the inverse of the predicted covariance) contributes in (17). When the query point is far from the learned region of KMP, the predicted covariance measures the uncertainty of the prediction itself. Namely, larger covariance corresponds to a less reliable predicted mean. These insights provide an alternative way to interpret (17).

In comparison, the local GP method [13], [21], [22] requires clustering the training dataset and explicitly estimating the distance from the query point to each cluster center. In contrast, the need for data clustering is eliminated in MiKMP. Note that in [13], [21], [22], the predicted uncertainty by GP is a scalar (or a diagonal matrix with identical diagonal entries), and therefore the output of the local GP method is essentially a linear combination of different GPs. Unlike local GP, the predicted covariance $\boldsymbol{\Sigma}^{(m)}$ in (16) is a full matrix capturing the correlations among the outputs. The covariance-weighted strategy in (17) resembles trajectory-GMM [23] and minimal intervention controller [24], [25]. The proposed mixture of predictions via the Gaussian product in (16) shares the spirit of some previous works, e.g., fusing predictions from different task frames [23], combining multiple optimal controllers [18], and integrating local HGP [10], but our primary goal here is to achieve computationally efficient KMP for adaptations.

V. Complexity Analysis

A. Computational Complexity of iKMP

We now analyze the time complexity of iKMP when coping with one desired point, where we assume that both 'removal' and 'addition' operations are involved. Let us start with the probabilistic reference trajectory $\mathbf{L}$ with size N and follow the KMP mean prediction in (1). Note that KMP predicts all outputs ($\mathcal{D}$-dimensional) together. To exclude the

[1]In practice, the segment lengths are not necessarily equal.

nearest datapoint from the reference trajectory, we need to determine $\widetilde{\mathbf{A}}, \widetilde{\mathbf{B}}, \widetilde{\mathbf{C}}, \widetilde{\mathbf{D}}$ in (11). This can be achieved by directly rearranging the elements in $(\mathbf{K}+\lambda_m\boldsymbol{\Sigma})^{-1}$, incurring a trivial computational cost. We emphasize that $(\mathbf{K}+\lambda_m\boldsymbol{\Sigma})^{-1}$ only needs to be computed once in advance. Since the dimensions of $\widetilde{\mathbf{A}}, \widetilde{\mathbf{B}}, \widetilde{\mathbf{C}}, \widetilde{\mathbf{D}}$ are $\mathcal{D}(N-1)\times\mathcal{D}(N-1)$, $\mathcal{D}(N-1)\times\mathcal{D}$, $\mathcal{D}\times\mathcal{D}(N-1)$, and $\mathcal{D}\times\mathcal{D}$, respectively, the time complexity of (12) is $\mathcal{O}(\mathcal{D}^3(N-1)^2)$ (due to matrix multiplications). For the case of adding a desired point to the updated reference trajectory, we can reuse $\mathbf{A}^{-1}$ in (12). The computational cost of $\boldsymbol{\Gamma}$ is $\mathcal{O}(\mathcal{D}^3(N-1)^2)$, where the dimensions of $\mathbf{E}$, $\mathbf{F}$, $\mathbf{G}$ are $\mathcal{D}(N-1)\times\mathcal{D}$, $\mathcal{D}\times\mathcal{D}(N-1)$, and $\mathcal{D}\times\mathcal{D}$, respectively. It can be shown that the time complexity of (15) is also $\mathcal{O}(\mathcal{D}^3(N-1)^2)$. Therefore, the time complexity of iKMP is $2\mathcal{O}(\mathcal{D}^3(N-1)^2)$.

In contrast, the computational cost of the expectation prediction in standard KMP is[2] $\mathcal{O}(\mathcal{D}^3N^3)$, where the length of the reference trajectory is still N after the nearest point is excluded and the new desired point is added.

B. Computational Complexity of MiKMP

Suppose that we divide the reference trajectory into M consecutive segments of equal length and that each segment has $\frac{N}{M}$ datapoints. For each segment, we learn a local model with a separate iKMP, and for each iKMP we have its time complexity as $2\mathcal{O}(\mathcal{D}^3(\frac{N}{M}-1)^2)$. To integrate the outputs from different iKMPs via (16), the covariance prediction in (1) is needed. If $\lambda_c \neq \lambda_m$, we can follow the same procedure in Section III to compute $(\mathbf{K}+\lambda_c\boldsymbol{\Sigma})^{-1}$. The additional time cost for each iKMP due to the covariance prediction is $2\mathcal{O}(\mathcal{D}^3(\frac{N}{M}-1)^2)$. As a result, each iKMP requires $4\mathcal{O}(\mathcal{D}^3(\frac{N}{M}-1)^2)$ time. For the Gaussian product in (17), the time cost is negligible since $\boldsymbol{\mu}^{(m)}$ is a $\mathcal{D}$-dimensional column vector and $\boldsymbol{\Sigma}^{(m)}$ is a $\mathcal{D}\times\mathcal{D}$ matrix. The ultimate computational cost for MiKMP is approximately $4M\cdot\mathcal{O}(\mathcal{D}^3(\frac{N}{M}-1)^2)$, which is smaller than $2\cdot\mathcal{O}(\mathcal{D}^3(N-1)^2)$ when $M>2$. Note that we perform both 'exclusion' and 'addition' operations for each desired point. Some desired points may require only the 'addition' update, thereby reducing the overall time cost of MiKMP.

VI. Evaluations

We perform evaluations on three tasks: a simulated letter writing task, an obstacle avoidance task with a real robot, and a real-world dual-arm transportation task. The simulation task runs on Python 3.9.18 on a 16-core 11-Gen Intel Core i9-11900 @ 2.50GHz processor, while the real robot tasks run on C++ on a 12-core 11-Gen Intel Core i5-11400 @ 2.60GHz processor. In the following evaluations, the Gaussian kernel $k(\mathbf{s}_i,\mathbf{s}_j)=\exp(-k_h||\mathbf{s}_i-\mathbf{s}_j||^2)$ is used for KMP-related methods.

A. Writting task

We consider the writing of the 2-D letters 'A' and 'G', where the demonstrations for both letters are obtained from [6]. The demonstrations consist of time inputs $\mathbf{s}=t$ and 2-D positions $\boldsymbol{\xi}=[x\ y]^T$. Five demonstrations are used for each letter. In addition to the evaluations of KMP, iKMP, and MiKMP, a mixture of KMPs is also tested, where a set of KMPs is used to learn different local models. ProMP [7] and NS-KMP [15] serve as baselines. For both letters, the length of the reference trajectory is $N=200$. The uniform division strategy is adopted for MiKMP.

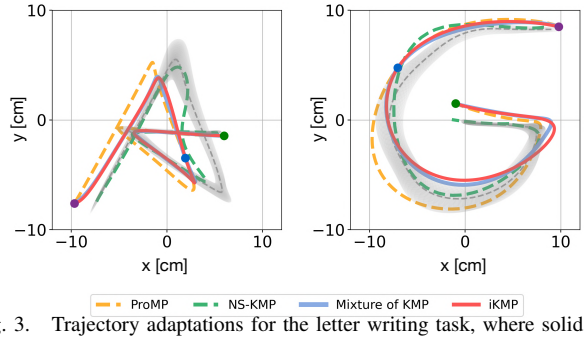

Fig. 3. Trajectory adaptations for the letter writing task, where solid dots indicate the desired points. The dashed grey curve and the shaded area represent the mean and standard deviation of the reference trajectory.

TABLE I
Computation Time for the Writing Task

Methods	Update Time* [s]	Prediction Time [s]
ProMP [7]	2.57×10^{-4}	3.86×10^{-5}
NS-KMP [15]	2.82×10^{-1}	/
KMP [9]	5.76×10^{-1}	9.10×10^{-4}
Mixture of KMP	1.42×10^{-1}	1.13×10^{-3}
iKMP	1.14×10^{-2}	8.86×10^{-4}
MiKMP	4.76×10^{-3}	1.11×10^{-3}

* *Here, we report the total update time required for all desired points.*

The adaptations using various methods are shown in Fig.3, where three desired points are imposed. For clarity, the trajectories generated by KMP and MiKMP are ignored, as their results are provided in Fig. 2. We can see that all methods except for NS-KMP are capable of generating trajectories that maintain the shape of demonstrations while passing through desired points.

The time costs for different methods are reported in Table I, corresponding to the average computational time for the two letters. For the mixture methods, four local models are used. For KMP and iKMP, the 'update time' corresponds to the calculation of $\boldsymbol{\alpha}=(\mathbf{K}+\lambda_m\boldsymbol{\Sigma})^{-1}\boldsymbol{\mu}$ and the 'prediction time' corresponds to the calculation of $\mathbf{k}^*\boldsymbol{\alpha}$. For the mixture versions (i.e., mixture of standard KMPs and instant KMPs), the 'prediction time' also includes the time required for predicting covariances, as well as the fusion computation in (17). For ProMP, the 'update time' refers to the time for weight update via Gaussian conditioning, while the 'prediction time' represents the time to predict a trajectory point for a single query. For NS-KMP, as it optimizes the entire trajectory in an iterative way, we take the whole optimization time as the 'update time'.

[2]While there are faster algorithms (e.g., Strassen's algorithm [26] shows a time complexity of $\mathcal{O}(n^{2.81})$, we here only consider naive matrix inverse.

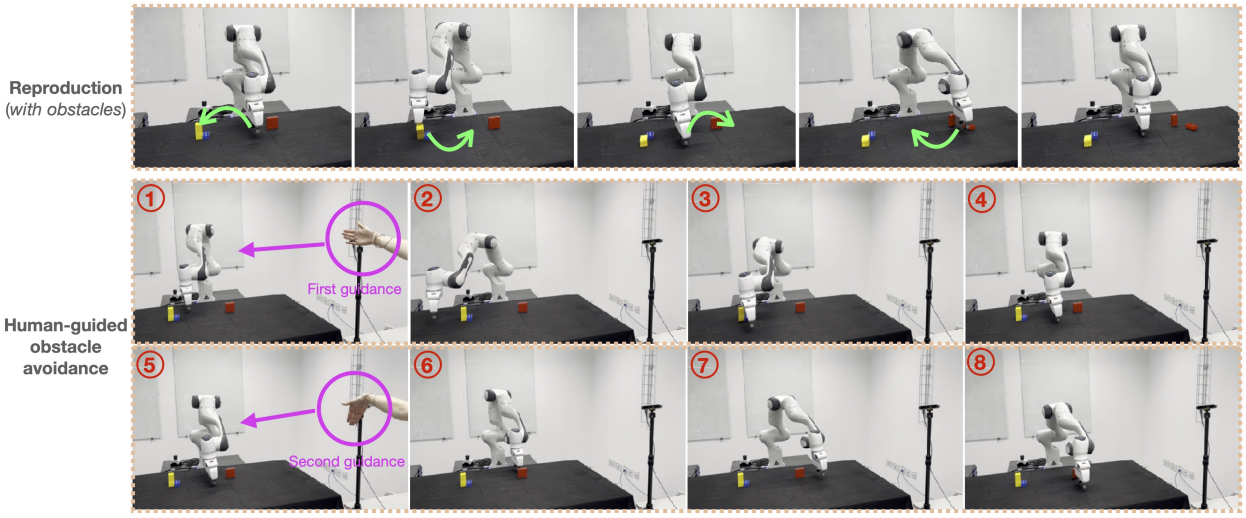

Fig. 4. Snapshots of the reproduction task (*first row*) and the human-guided obstacle avoidance task (*second and third rows*) using MiKMP. Since these obstacles are not present during the demonstration collection stage, the reproduction task collides with them.

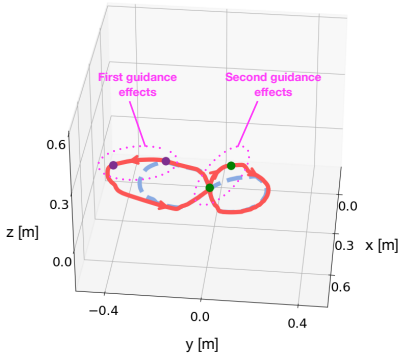

Fig. 5. Evaluations of MiKMP on the obstacle avoidance task, where the reproduction and adaptation trajectories are displayed by the blue and red curves, respectively. The solid dots, corresponding to the desired points, are determined by the user's hand gesture.

TABLE II

COMPUTATION TIME FOR THE OBSTACLE AVOIDANCE TASK

Methods	**Update Time*[s]**	**Prediction Time [s]**
KMP	2.89	$\mathbf{2.11} \times 10^{-4}$
iKMP	2.27×10^{-1}	2.13×10^{-4}
MiKMP	2.69×10^{-2}	2.38×10^{-3}
MiKMP (*Parallel*)	$\mathbf{1.03} \times 10^{-2}$	1.29×10^{-3}

* *The total update time for a single human-guidance event, involving two desired points, is reported.*

It can be seen from Table I that the mixture treatment (for both the mixture of KMP and the mixture of iKMP) reduces the 'update time' at the cost of a negligible increase in prediction time. Although the time costs for the nonparametric method MiKMP ($M = 4$) are higher than those of the parametric method ProMP, they remain favourable at the millisecond level.

Note that, as shown in [15], NS-KMP is combined with an external optimizer to minimize an objective function defined by task requirements (e.g., obstacle avoidance), where a large number of iterations are needed, each with computational complexity $\mathcal{O}(\mathcal{D}^3 N^2)$. In our evaluations, we define a different objective function that calculates the distance between the generated trajectory and the three desired points. In addition to the iterative search, the time NS-KMP takes also largely depends on the sampling ranges and relevant hyperparameters of the optimizer, which becomes cumbersome in the writing tasks with three desired points. Indeed, generating a smooth trajectory to pass through three desired points is challenging, and the results show that NS-KMP typically requires more than $0.2s$, which is noticeably longer than the proposed solutions here.

B. Obstacle Avoidance Task

We first teach the robot five demonstrations of plotting a trajectory with the shape '∞'. Each demonstration consists of time inputs $\mathbf{s} = t$ and the robot's end-effector positions $\boldsymbol{\xi} = [x\ y\ z]^T$. The length of the reference trajectory is $N = 100$. Later, unseen obstacles colliding with the demonstrations are placed, requiring the robot to adapt its trajectory to avoid the collisions. Specifically, we use a depth camera to detect the type of the user's hand posture (up, down, left, right), which is subsequently used to determine desired points for the robot on the fly. The idea of adapting the robot's trajectory towards new desired points so as to avoid obstacles has been exploited in [9]. Here, we focus on verifying the effectiveness of our methods in the scenario demanding online adaptations. We use the detection method in [27] for hand recognition.

The snapshots of the reproduction task with unseen obstacles via MiKMP ($M = 4$) are shown in the *first row* of Fig. 4, while the adaptation task for obstacle avoidance using MiKMP is shown in the *second and third rows*. For the adaptation task, human guidance is introduced twice, with two desired points defined for each guidance. Assuming that the user performs a hand gesture at time t and the

Fig. 6. Snapshots of demonstration collection (*first row*) and the transportation of two baskets (*second and third rows*) using MiKMP. Note that only the transportation of the top basket is demonstrated, while the bottom basket requires adaptation.

corresponding robot state is $\mathbf{p}_t$, the first desired point will be $\bar{\mathbf{s}}_1 = t$ and $\bar{\boldsymbol{\mu}}_1 = \mathbf{p}_t$, ensuring that the adapted trajectory moves smoothly from the current position. The second desired point is defined as $\bar{\mathbf{s}}_2 = t + \delta_t$ and $\bar{\boldsymbol{\mu}}_2 = \mathbf{p}_t + \Delta_p$, where δ_t denotes the time offset and Δ_p represents the desired adjustment for the robot's position, determined by the type of user's hand gesture. For both desired points, we have $\bar{\boldsymbol{\Sigma}}_1 = \bar{\boldsymbol{\Sigma}}_2 = 10^{-9}\mathbf{I}$ to ensure accurate adaptations. The hyperparameters for the MiKMP are $k_h = 0.2$ and $\lambda_m = \lambda_c = 2$. The real robot's trajectories, with (red curve) and without (blue curve) adaptations, are presented in Fig. 5.

The computational costs are summarized in Table II, where MiKMP with multithreading (i.e., *parallel computing*) is also implemented since different iKMPs run independently in the framework of MiKMP (see Fig. 1). The results again evidence that iKMP is faster than KMP, the mixture treatment and parallel computing further boost the efficiency.

C. Dual-Arm Transportation Task

The objective of this task is to transport two baskets from one location to the other. Snapshots of collecting a demonstration are provided in Fig. 6 (*first row*). Following this procedure, five demonstrations for both robotic arms are collected. For the right robotic arm, the demonstrations consist of time inputs $\mathbf{s} = t$ and its 7-D end-effector poses $\boldsymbol{\xi} = [\mathbf{p}^{r^T}\ \mathbf{q}^{r^T}]^T$ (i.e., 3-D Cartesian position $\mathbf{p}^r$ and 4-D quaternion $\mathbf{q}^r$). For the left robotic arm, the demonstrations comprise the right arm's pose (7-D) and its end-effector pose (7-D), i.e., $\mathbf{s} = [\mathbf{p}^{r^T}\ \mathbf{q}^{r^T}]^T$ and $\boldsymbol{\xi} = [\mathbf{p}^{l^T}\ \mathbf{q}^{l^T}]^T$. In this way, the left arm takes the right arm's state as input and synchronizes its actions accordingly. This task can be transferred to human-robot collaboration (e.g., hand-over task), where the user's hand pose can be interpreted as the right robotic arm's state. The length of the reference trajectory is $N = 100$.

Note that we only show both robots how to transport the top basket; they need to learn to adapt to the bottom basket, as it requires new picking and placing locations. For each transportation, we define three desired points: one for the starting location of the robot, the second for the picking location, and the third for the placing location. For the right robotic arm, the input and output of all desired points are time and the desired 7-D pose for the robot. For the left arm, the input and output of all desired points are the 7-D pose of the right arm and the desired 7-D pose for itself. Note that, as discussed in [9], [15], it is difficult to employ ProMP for such a task due to the large number of basis functions and the corresponding hyperparameters (e.g., centers and bandwidths for Gaussian basis functions).

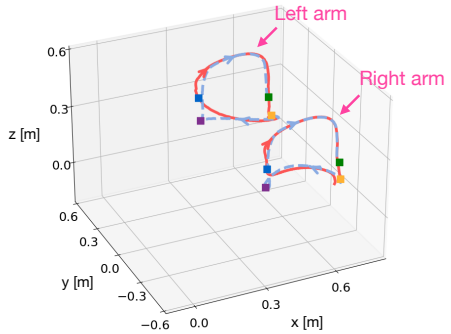

Fig. 7. Evaluations of MiKMP in the dual-arm transportation task, where the real robotic trajectories for transporting the top (*red curves*) and bottom (*blue curves*) baskets are plotted. The solid squares indicate the picking and placing locations, where —picking the top basket; —placing the top basket; —picking the bottom basket; —placing the bottom basket.

The real robot trajectories of the transportation tasks using MiKMP ($M = 4$) are plotted in Fig. 7, where the red curves show the transportation of the top basket, while the blue curves correspond to the bottom basket. The hyperparameters of the MiKMP used for the right robotic arm are $k_h = 0.0015$ and $\lambda_m = \lambda_c = 10$, while for the left arm these parameters are $k_h = 0.04$ and $\lambda_m = \lambda_c = 0.1$. These parameters are chosen empirically.

TABLE III
COMPUTATION TIME FOR THE TRANSPORTATION TASK
(RIGHT ROBOTIC ARM, TIME INPUT)

Methods	Update Time* [s]	Prediction Time [s]
KMP	13.44	$\mathbf{2.78} \times 10^{-4}$
iKMP	7.09×10^{-1}	2.83×10^{-4}
MiKMP	9.38×10^{-2}	2.58×10^{-3}
MiKMP (*Parallel*)	$\mathbf{3.98} \times 10^{-2}$	2.48×10^{-3}

* *The total update time required for the entire task is reported.*

TABLE IV
COMPUTATION TIME FOR THE TRANSPORTATION TASK
(LEFT ROBOTIC ARM, 7-D INPUT)

Methods	Update Time* [s]	Prediction Time [s]
KMP	13.17	$\mathbf{2.92} \times 10^{-4}$
iKMP	4.82×10^{-1}	2.93×10^{-4}
MiKMP	9.48×10^{-2}	3.17×10^{-3}
MiKMP (*Parallel*)	$\mathbf{5.20} \times 10^{-2}$	2.57×10^{-3}

* *The total update time for the entire task is presented.*

The snapshots of transporting both baskets using MiKMP are shown in Fig. 6 (*second and third rows*), showing that MiKMP indeed accomplishes the task. As the left robotic arm moves in response to the right arm, we also introduce perturbations to the right arm. The results show that the left robot can make prompt adjustments whenever external perturbations are applied to the right robotic arm, ensuring the task is accomplished. See the accompanying video for the perturbation evaluations.

We test the time needed for different methods and report the results in Table III and Table IV. Again, it can be seen that our proposed solutions largely improve computational efficiency compared to standard KMP.

VII. CONCLUSIONS

We developed computationally efficient nonparametric imitation learning solutions on top of KMP. Evaluations on the 2-D writing task, 3-D obstacle avoidance task, and the dual-arm transportation task (involving 7-D input and 7-D output) showed that our approach significantly reduced computational time compared with standard KMP. Notably, the writing task demonstrated that our method showed comparable performance to the parametric baseline. In future work, we will extend our solution to address constrained learning problems, such as orientation and stillness learning, and various hard constraints imposed by tasks and robots.

REFERENCES

[1] 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.

[2] 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.

[3] Y. Huang and D. G. Caldwell, "A linearly constrained nonparametric framework for imitation learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 4400–4406.

[4] R. A. Shyam, P. Lightbody, G. Das, P. Liu, S. Gomez-Gonzalez, and G. Neumann, "Improving local trajectory optimisation using probabilistic movement primitives," in *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, 2019, pp. 2666–2671.

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

[6] S. Calinon and D. Lee, "Learning control," in *Humanoid robotics: A reference*. Springer, 2017, pp. 1–52.

[7] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, "Probabilistic movement primitives," in *Advances in neural information processing systems*, 2013, pp. 2616–2624.

[8] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[9] 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.

[10] M. Schneider and W. Ertel, "Robot learning by demonstration with local gaussian process regression," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010, pp. 255–260.

[11] E. Snelson and Z. Ghahramani, "Sparse gaussian processes using pseudo-inputs," *Advances in neural information processing systems*, vol. 18, 2005.

[12] S. Van Vaerenbergh, J. Via, and I. Santamaría, "A sliding-window kernel rls algorithm and its application to nonlinear channel identification," in *IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, vol. 5, 2006, pp. V–V.

[13] D. Nguyen-Tuong, M. Seeger, and J. Peters, "Model learning with local gaussian process regression," *Advanced Robotics*, vol. 23, no. 15, pp. 2015–2034, 2009.

[14] M. Seeger *et al.*, "Low rank updates for the cholesky decomposition," *University of California at Berkeley, Tech. Rep*, 2004.

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

[16] 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.

[17] S. Wu, Y. Wang, and Y. Huang, "Autolfd: Closing the loop for learning from demonstrations," *IEEE Transactions on Automation Science and Engineering*, vol. 22, pp. 11 124–11 138, 2025.

[18] J. Silvério, Y. Huang, F. Abu-Dakka, L. Rozo, and D. Caldwell, "Uncertainty-aware imitation learning using kernelized movement primitives," in *International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 90–97.

[19] S. Van Vaerenbergh, I. Santamaría, W. Liu, and J. C. Príncipe, "Fixed-budget kernel recursive least-squares," in *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2010, pp. 1882–1885.

[20] K. B. Petersen, M. S. Pedersen *et al.*, "The matrix cookbook," *Technical University of Denmark*, vol. 7, no. 15, p. 510, 2008.

[21] D. Nguyen-Tuong, J. Peters, and M. Seeger, "Local gaussian process regression for real time online model learning," *Advances in neural information processing systems*, vol. 21, 2008.

[22] B. Wilcox and M. C. Yip, "Solar-gp: Sparse online locally adaptive regression using gaussian processes for bayesian robot model learning and control," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2832–2839, 2020.

[23] S. Calinon, "A tutorial on task-parameterized movement learning and retrieval," *Intelligent Service Robotics*, vol. 9, no. 1, pp. 1–29, 2016.

[24] J. R. Medina, D. Lee, and S. Hirche, "Risk-sensitive optimal feedback control for haptic assistance," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2012, pp. 1025–1031.

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

[26] V. Strassen, "Gaussian elimination is not optimal," *Numerische mathematik*, vol. 13, no. 4, pp. 354–356, 1969.

[27] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee *et al.*, "Mediapipe: A framework for building perception pipelines," *arXiv preprint arXiv:1906.08172*, 2019.