2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Learning and Sequencing of Object-Centric Manipulation Skills for Industrial Tasks[1]

Leonel Rozo*, Meng Guo*, Andras G. Kupcsik*, Marco Todescato, Philipp Schillinger,
Markus Giftthaler, Matthias Ochs, Markus Spies, Nicolai Waniek, Patrick Kesper, Mathias Bürger

Abstract— **Enabling robots to quickly learn manipulation skills is an important, yet challenging problem. Such manipulation skills should be flexible, e.g., be able adapt to the current workspace configuration. Furthermore, to accomplish complex manipulation tasks, robots should be able to sequence several skills and adapt them to changing situations. In this work, we propose a rapid robot skill-sequencing algorithm, where the skills are encoded by object-centric hidden semi-Markov models. The learned skill models can encode multimodal (temporal and spatial) trajectory distributions. This approach significantly reduces manual modeling efforts, while ensuring a high degree of flexibility and re-usability of learned skills. Given a task goal and a set of generic skills, our framework computes smooth transitions between skill instances. To compute the corresponding optimal end-effector trajectory in task space we rely on Riemannian optimal controller. We demonstrate this approach on a 7 DoF robot arm for industrial assembly tasks.**

I. INTRODUCTION

Deploying service robots in highly flexible manufacturing sites is promising, but also challenging [1]. The challenges arise in different sub-fields of robotics, e.g., perception [2], motion planning [3], mapping and navigation [4], or human-robot interaction [5]. In this work, we tackle two specific problems, namely, flexible motion skills generation and skills sequencing in the context of industrial tasks, with an emphasis on assembly settings. First, it is impossible for robot manufacturers to pre-program *all* robot capabilities (referred to as *skills*) that end users may require. To avoid inquiring engineers whenever a new skill is needed, it is crucial to provide an easy and efficient method with which laymen can teach the robot new skills. Simply recording and replaying a demonstrated trajectory is often insufficient, because changes in the environment, e.g., varying robot and/or object poses, would render any attempt unsuccessful. In other words, the robot needs to recognize the intentions behind these demonstrations and thus generalize over unforeseen situations.

Many learning-from-demonstration (LfD) frameworks have shown great improvements in this aspect. Compared to hard-coded alternatives, they embed extracted knowledge into probabilistic models. Examples are probabilistic movement primitives (ProMPs) [6], Stable Estimators of Dynamical Systems (SEDS) [7], Task-Parameterized Gaussian Mixture Models (TP-GMMs) [8], and more recently, Kernelized MPs (KMPs) [9] and Conditional Neural MPs (CNMPs) [10]. However, most of these approaches train models specifically for each skill instantiation, with the exception of CNMPs, which is able to encode multiple modes of operation for the same skill. For most of the aforementioned models, "grasp the object from the top" and "grasp the object from the side" are usually treated as two different skills, and thus, different models are trained. This not only greatly decreases the teaching efficiency, but also significantly limits the *reusability* of each skill. Instead, we consider object-centric skills that represent robot end-effector motions relative to objects of interest for the task space. To encode and generate end-effector pose trajectories, we exploit Riemannian-manifold theory to compute the necessary statistics and retrieve smooth control references. Recent work in robot learning, control and optimization showed the efficiency and stability of Riemannian methods [11], [12].

Additionally, several skills often need to be performed in sequence to accomplish tasks with increased complexity. In this work, we assume such a sequence is given by the user and our focus is on the adaptation of each skill to the sequence and to the varying configurations. The problem of finding the right sequence is commonly referred to as the task planning problem [13]. Logic-based planning frameworks such as PDDL [14] gained popularity due to close resemblance to human reasoning. However, manual definition of the planning model, such as pre-conditions and effects of all skills, quickly becomes impractical due to the large variation of skills in different applications.

In this paper we propose a generic motion planning framework for sequencing manipulation skills that are learned from demonstration. First, to teach a *general* skill, only few human demonstrations are needed with different object configurations. These demonstrations are used to train an object-centric model for each skill, which builds local models of the demonstrations from the perspective of different coordinate systems. Moreover, given a sequence of skills to execute, a complete model is constructed by cascading the local models with updated parameters. Lastly, during execution, this model is used to compute the most-likely reference trajectory for the robot under various workspace configurations. Our method significantly reduces human modeling efforts, while ensuring high-degree flexibility and re-usability of learned skills.

The main contribution of this paper is twofold: (i) we present a novel algorithm for the sequencing of several general skills to fulfill a given task; (ii) and propose a skill-sequencing method that finds the most-likely reference trajectory given only the initial system state and the desired task goal. Our framework builds on a Riemannian-manifold

* Equal contribution
[1]Bosch Center for Artificial Intelligence, Renningen, Germany `<first name(s)>.<last name>@de.bosch.com`

formulation to provide robust learning and optimal control, overcoming inaccuracy and stability issues that arise when using Euclidean approximations.

The remainder of this paper is organized as follows. Section II reviews the state of the art in motion primitives and skills sequencing. Section III presents some preliminaries on TP-GMMs and Riemannian geometry, essential tools in our work. Section IV details the considered problem. Section V contains the main contributions. Experimental results are presented in Section VI. Section VII concludes the work.

II. Related Work

Learning by demonstration is an intuitive and natural way to transfer human skills to robots, which recently gained much attention [15]. Gaussian Mixture Models (GMMs) provide an elegant probabilistic representation of motion skills. For instance, the work by Niekum *et al.* [16] shows how to use them to extract important features from only few human demonstrations of particular skills. Successful applications can be found in humanoids [15], human-robot collaborative manipulation [17] and robot motion planning [18]. Furthermore, task-parameterized GMMs (TP-GMMs) [8] provide a powerful extension to GMMs by incorporating observations from the perspective of different frames of reference. This allows the robot to automatically adapt movements to new situations and has shown reliable performance in human-robot collaborative transportation [17] and robot bimanual sweeping [19]. However, most of the above approaches focus on learning a specific TP-GMM model for each single skill, e.g., "holding the cube above the cup" [8] or "transporting the object" [17]. In contrast, we propose to learn one TP-GMM model for a general class of skills (without explicitly distinguishing them) and choose the particular instantiation of each skill during run-time, depending on the given high-level plan and task goal.

Given a set of skills, the next problem is how to combine them for successful execution of complex manipulation tasks. Researchers have mainly used either LfD [20], [21], [22] or reinforcement learning (RL) [23], [24] to master skills sequencing, most of them employing dynamic movement primitives (DMPs) as skill representation. Manschitz *et al.* [20] learn a sequence graph of skills from kinesthetic demonstrations, where a classifier drives transitions between skills. The authors extend this approach for bimanual settings [21], where the task is represented by a set of concurrent sequence graphs of motion primitives. Entry and exit probabilities determine the transition between consecutive skills conditioned on the environment state. Pastor *et al.* [22] learn DMPs along with a distribution of sensory patterns. Skills sequencing is achieved by choosing pairs of DMPs whose initial and final sensory patterns closely match. Our work distinguishes in that our skill representation encodes the diverse effects of robot actions on objects, which may differ according to the particular instantiation of the skill. Similarly to [22], our method exploits the distribution of observations to build a task model by cascading different skills as a function of their similarity in a Kullback-Leibler sense.

Gräve and Behnke [23] combine LfD and RL to learn a sequence of skills in an active learning setting. In [24], a modified PI^2 method adapts the shape and attractor of several learned DMPs to smoothly sequence them. A similar approach is proposed in [25] to sequence DMPs in bimanual manipulation. In [26], the sequencing problem is investigated from a hierarchical RL perspective, where each skill is represented by a general control policy. The work by Mülling *et al.* [27] proposes a gating network that activates the appropriate skill among a set of learned DMPs for table tennis. Lastly, Kroemer *et al.* [28] use model-based RL to learn a high-level policy that sequences learned motion skills. Most of the above approaches require human demonstrations for the complete task. Moreover, RL is used to adapt the skill parameters to make skills sequencing possible. In contrast, our approach requires only demonstrations on the level of *individual* skills. RL adaptation of skill parameters for sequencing is unnecessary as our method builds a complete model of the task that considers the different skill instantiations and exploits predictive models of the skill effects.

III. Preliminaries

We briefly present some preliminary results in robot skill learning, in particular, TP-GMMs, hidden semi-Markov models (HSMMs), and Riemannian manifolds.

A. TP-GMMs

The basic idea of LfD is to fit a prescribed skill model such as GMMs to a handful of demonstrations. We assume we are given M demonstrations, each of which contains T_m data points for a dataset of $N = \sum_m T_m$ total observations $\boldsymbol{\xi} = \{\boldsymbol{\xi}_t\}_{t=1}^N$, where $\boldsymbol{\xi}_t \in \mathbb{R}^d$. Also, we assume the same demonstrations are recorded from the perspective of P different coordinate systems (given by the task parameters, such as, objects of interest). One common way to obtain such data is to transform the demonstrations from a static global frame to frame p by $\boldsymbol{\xi}_t^{(p)} = \boldsymbol{A}^{(p)^{-1}}(\boldsymbol{\xi}_t - \boldsymbol{b}^{(p)})$. Here, $\{(\boldsymbol{b}^{(p)}, \boldsymbol{A}^{(p)})\}_{p=1}^P$ is the translation and rotation of frame p w.r.t. the world frame. Then, a TP-GMM is described by the model parameters $\{\pi_k, \{\boldsymbol{\mu}_k^{(p)}, \boldsymbol{\Sigma}_k^{(p)}\}_{p=1}^P\}_{k=1}^K$ where K represents the number of Gaussian components in the mixture model, π_k is the prior probability of each component, and $\{\boldsymbol{\mu}_k^{(p)}, \boldsymbol{\Sigma}_k^{(p)}\}_{p=1}^P$ are the parameters of the k-th Gaussian component within frame p. Differently from standard GMM, the mixture model above can not be learned independently for each frame. Indeed, the mixing coefficients π_k are shared by all frames and the k-th component in frame p must map to the corresponding k-th component in the global frame. Expectation-Maximization (EM) [8] is a well-established method to learn such models. Task parameterization of GMMs incorporates observations from the perspective of different frames of reference, thus allowing the robot to automatically adapt its motion to new situations.

Once learned, the TP-GMM can be used during execution to reproduce a trajectory for the learned skill. Namely, given the observed frames $\{\boldsymbol{b}^{(p)}, \boldsymbol{A}^{(p)}\}_{p=1}^P$, the learned TP-GMM is converted into one single GMM with parameters

$\{\pi_k, (\hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k)\}_{k=1}^K$, by multiplying the affine-transformed Gaussian components across different frames, as follows

$$\hat{\boldsymbol{\Sigma}}_k = \left[\sum_{p=1}^{P} \left(\hat{\boldsymbol{\Sigma}}_k^{(p)}\right)^{-1}\right]^{-1}, \hat{\boldsymbol{\mu}}_k = \hat{\boldsymbol{\Sigma}}_k \left[\sum_{p=1}^{P} \left(\hat{\boldsymbol{\Sigma}}_k^{(p)}\right)^{-1} \hat{\boldsymbol{\mu}}_k^{(p)}\right], \quad (1)$$

where the parameters of the updated Gaussian at each frame p are computed as $\hat{\boldsymbol{\mu}}_k^{(p)} = \boldsymbol{A}^{(p)}\boldsymbol{\mu}_k^{(p)} + \boldsymbol{b}^{(p)}$ and $\hat{\boldsymbol{\Sigma}}_k^{(p)} = \boldsymbol{A}^{(p)}\boldsymbol{\Sigma}_k^{(p)}\boldsymbol{A}^{(p)^{\mathsf{T}}}$. While the task parameters may vary over time, we dropped the time index for the sake of notation.

B. HSMMs

Hidden semi-Markov Models (HSMMs) extend standard hidden Markov Models (HMMs) by embedding temporal information of the underlying stochastic process. That is, while in HMM the probability of transitioning to the next state depends only on the current state, in HSMM this transition also depends on the elapsed time since the state was entered. HSMMs have been successfully applied, in combination with TP-GMMs, for robot skill encoding to learn spatio-temporal features of the demonstrations [29]. More specifically, a task-parameterized HSMM (TP-HSMM) model is defined as:

$$\boldsymbol{\Theta} = \left\{\{a_{hk}\}_{h=1}^K, (\mu_k^D, \sigma_k^D), \pi_k, \{(\boldsymbol{\mu}_k^{(p)}, \boldsymbol{\Sigma}_k^{(p)})\}_{p=1}^P\right\}_{k=1}^K,$$

where a_{hk} is the transition probability from state h to k; (μ_k^D, σ_k^D) describe the Gaussian distributions for the duration of state k, i.e., the probability of staying in state k for a certain number of consecutive steps; $\{\pi_k, \{\boldsymbol{\mu}_k^{(p)}, \boldsymbol{\Sigma}_k^{(p)}\}_{p=1}^P\}_{k=1}^K$ equal the TP-GMM introduced earlier, representing the observation probability corresponding to state k. Note that in our HSMM the number of states corresponds to the number of Gaussian components in the "attached" TP-GMM.

Given a certain (partial) sequence of observed data points $\{\boldsymbol{\xi}_\ell\}_{\ell=1}^t$, assume that the associated sequence of states in $\boldsymbol{\Theta}$ is given by $\boldsymbol{s}_t = s_1 s_2 \cdots s_t$. As shown in [29], the probability of data point $\boldsymbol{\xi}_t$ belonging to state k (i.e., $s_t = k$) is given by the *forward* variable $\alpha_t(k) = p(s_t = k, \{\boldsymbol{\xi}_\ell\}_{\ell=1}^t)$:

$$\alpha_t(k) = \sum_{\tau=1}^{t-1} \sum_{h=1}^{K} \alpha_{t-\tau}(h) a_{hk} \mathcal{N}(\tau|\mu_k^D, \sigma_k^D)\, o_\tau^t, \quad (2)$$

where $o_\tau^t = \prod_{\ell=t-\tau+1}^{t} \mathcal{N}(\boldsymbol{\xi}_\ell|\hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k)$ is the emission probability and $(\hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k)$ are derived from (1) given the task parameters. Furthermore, the same forward variable can also be used during reproduction to predict future steps until T_m.

In this case however, since future observations are not available, only transition and duration information are used as explained by [30], i.e., by setting $\mathcal{N}(\boldsymbol{\xi}_\ell|\hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k) = 1$ for all k and $\ell > t$ in (2). At last, the sequence of the most-likely states $\boldsymbol{s}_{T_m}^\star = s_1^\star s_2^\star \cdots s_{T_m}^\star$ is determined by choosing $s_t^\star = \arg\max_k \alpha_t(k)$, $\forall 1 \leqslant t \leqslant T_m$.

C. Riemannian Manifolds

As the robot motion skills are learned from and reproduce time-varying poses of the end-effector, classical Euclidean-based methods are inadequate for processing such data, as they rely on rough approximations to account for the constraints imposed by orientation representation such as quaternions. These approximations may lead to inaccurate skill models or unstable controllers. We instead endow the robot task space with a Riemannian manifold $\mathcal{M}$ [11]. Briefly, for each point $\boldsymbol{x}$ in the manifold $\mathcal{M}$, there exists a tangent space $\mathcal{T}_{\boldsymbol{x}}\mathcal{M}$. This allows us to carry out Euclidean operations locally, while being geometrically consistent with manifold constraints.

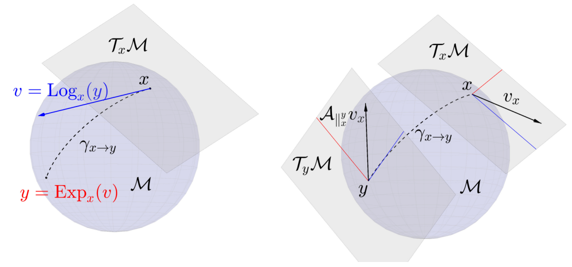

Fig. 1: **Left:** the illustration of the Log and Exp maps with geodesic $\gamma_{x\to y}$. Note that $\|v\|_2 = \|\gamma_{x\to y}\|_2$. **Right:** illustration of the parallel transport operation. v_x is a vector defined in the tangent space of x, while the parallel transported vector $\mathcal{A}_{\|_x^y} v_x$ will lie in the tangent space of y and is considered parallel to v_x.

We use exponential and logarithmic maps to map points between $\mathcal{T}_{\boldsymbol{x}}\mathcal{M}$ and $\mathcal{M}$ (Fig. 1 left). The exponential map $\text{Exp}_{\boldsymbol{x}} : \mathcal{T}_{\boldsymbol{x}}\mathcal{M} \to \mathcal{M}$ maps a point in the tangent space of point $\boldsymbol{x}$ to a point on the manifold, while maintaining the geodesic distance. The inverse operation is called the logarithmic map $\text{Log}_{\boldsymbol{x}} : \mathcal{M} \to \mathcal{T}_{\boldsymbol{x}}\mathcal{M}$. Another useful operation is the parallel transport $\mathcal{A}_{\|_{\boldsymbol{x}}^{\boldsymbol{y}}} : \mathcal{T}_{\boldsymbol{x}}\mathcal{M} \to \mathcal{T}_{\boldsymbol{y}}\mathcal{M}$, which moves elements between tangent spaces without introducing distortion (Fig. 1 right). The exact form of the aforementioned operations depend on the Riemannian metric associated to the manifold, which in our case corresponds to the formulations in [11].

In this paper we exploit Riemannian manifolds to: **(a)** properly compute statistics over $\mathcal{M}$ using Riemannian normal distributions that encode full end-effector motion patterns [11]; and **(b)** retrieve a smooth reference trajectory corresponding to the task plan (i.e., sequenced skills) using Riemannian optimal control, as proposed in Section V-A.2.

IV. Problem Description

Consider a multi-DoF robotic arm, whose end-effector has state $\boldsymbol{x}_{\mathsf{e}} \in \mathbb{R}^3 \times \mathcal{S}^3 \times \mathbb{R}^1$ (describing the Cartesian position, orientation quaternion and gripper state), that operates within a static and known workspace. Also, within the reach of the arm, there are objects of interest denoted by $\mathsf{O} = \{\mathsf{o}_1, \mathsf{o}_2, \cdots, \mathsf{o}_J\}$, each of which has state $\boldsymbol{x}_{\mathsf{o}_j} \in \mathbb{R}^3 \times \mathcal{S}^3$. For simplicity, the overall system state is denoted by $\boldsymbol{x} = \{\boldsymbol{x}_{\mathsf{e}}, \{\boldsymbol{x}_{\mathsf{o}_j}, \forall \mathsf{o}_j \in \mathsf{O}\}\}$.

Within this setup, an operator performs several kinesthetic demonstrations on the arm to manipulate one or several objects for certain manipulation skills. Denote by $\mathsf{A} = \{\mathsf{a}_1, \mathsf{a}_2, \cdots, \mathsf{a}_H\}$ the set of demonstrated skills. Moreover, for skill $\mathsf{a} \in \mathsf{A}$, the set of objects involved is given by O_{a} and the set of available demonstrations is denoted by D_{a}. Note that all demonstrations follow the object-centric

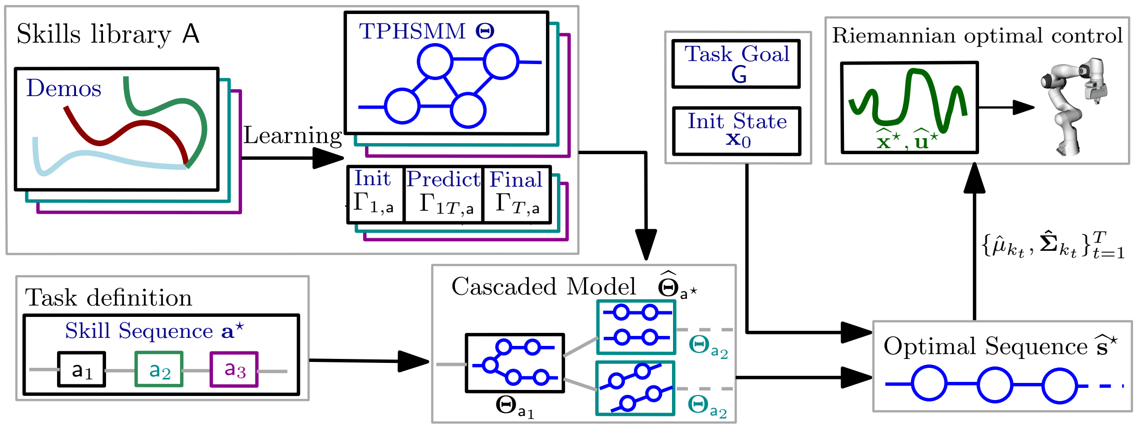
Skills library A
Demos
Learning
TPHSMM $\boldsymbol{\Theta}$
Init $\Gamma_{1,\mathsf{a}}$
Predict $\Gamma_{1T,\mathsf{a}}$
Final $\Gamma_{T,\mathsf{a}}$
Task Goal G
Init State $\mathbf{x}_0$
Riemannian optimal control
$\widehat{\mathbf{x}}^\star, \widehat{\mathbf{u}}^\star$
$\{\hat{\mu}_{k_t}, \hat{\boldsymbol{\Sigma}}_{k_t}\}_{t=1}^T$
Task definition
Skill Sequence $\mathbf{a}^\star$
a_1
a_2
a_3
Cascaded Model $\widehat{\boldsymbol{\Theta}}_{\mathsf{a}^\star}$
$\boldsymbol{\Theta}_{\mathsf{a}_1}$
$\boldsymbol{\Theta}_{\mathsf{a}_2}$
$\boldsymbol{\Theta}_{\mathsf{a}_2}$
Optimal Sequence $\widehat{\mathbf{s}}^\star$

Fig. 2: Overall diagram of the proposed method. From multiple kinesthetic demonstrations we encode object-centric skill models into TP-HSMMs and learn precondition/prediction/effect models $\boldsymbol{\Gamma}_{\mathsf{a}}$. Given a high-level task definition, we cascade the learned skill models into a joint model $\widehat{\boldsymbol{\Theta}}_{\mathsf{a}^\star}$. Based on the desired goal G of the task and the initial state of the environment, we find the optimal state-sequence $\widehat{\mathbf{s}}^\star$ using a modified Viterbi algorithm. Finally, to generate the corresponding reference trajectory we rely on a Riemannian optimal controller.

structure introduced in Section III-A, i.e., they are recorded from multiple frames, often associated to the pose of the objects in O_{a}. For example, the skill "insert the peg in the cylinder" involves the objects "peg" and "cylinder", and the associated demonstrations are recorded from the robot, the "peg" and the "cylinder" frames. Note that one general skill can include several execution instances, e.g., "pick the peg" includes "pick the peg from top, left or right", and "drop the peg" comprises "drop the peg inwards or outwards".

The manipulation tasks we consider consist of a *given* sequence of skills $\mathbf{a}^\star$ chosen from the demonstrated skills A. For example, an insertion task involves "pick the cap, re-orient the cap, pick the cap again and the insert the cap". In the end of the task, a goal configuration G is reached as the desired final state of the system, including the robot and the objects. The considered problem is as follows:

Problem 1: Given a set of demonstrated skills A, the desired skill sequence $\mathbf{a}^\star$ and the goal G, the objective is twofold: **(a)** to reproduce each skill in $\mathbf{a}^\star$ for a given goal state; and **(b)** to subsequently reproduce the sequence $\mathbf{a}^\star$ to maximize the success rate of achieving the goal G. ■

V. Learning and Sequencing of Flexible Skills

In this section, we present the main building blocks of the proposed framework, regarding the two objectives above. In Figure 2 we show the diagram of our proposed approach.

A. Single Skill Reproduction

Consider one demonstrated skill $\mathsf{a} \in \mathsf{A}$, associated with the set of demonstrations $\mathsf{D}_{\mathsf{a}} = \{\boldsymbol{\xi}_t\}_{t=1}^N$, recorded in P frames. Note that such frames are directly attached to the objects in O_{a}. As mentioned in Section III-B, given a properly chosen number of components K, the TP-HSMM model $\boldsymbol{\Theta}_{\mathsf{a}}$ abstracting the spatio-temporal features of trajectories related to skill a, can be learned using an EM-like algorithm. It is worth emphasizing that, conversely to most existing work that only treats each instance of the same skill separately, we construct one model for the general skill. To give an example, Fig. 3 shows an illustration of an HSMM for "pick the peg" skill that contains 10 demonstrations for "pick from top" and "pick from side". The learned HSMM in the global frame has a single initial state from which two branches encode the two different instances of the same "pick" skill.

*1) **Optimal state sequence**:* Consider now that a desired final observation of the robot state is given as $\boldsymbol{\xi}_T$, where T is the skill time horizon (e.g. the average length over the demonstrations). Moreover, the initial robot state is observed as $\boldsymbol{\xi}_1$. We firstly address the following sub-problem:

Problem 2: Given the learned model $\boldsymbol{\Theta}_{\mathsf{a}}$, construct the most-likely state sequence $\boldsymbol{s}_T^\star$ given *only* $\boldsymbol{\xi}_1$ and $\boldsymbol{\xi}_T$. ■

The approach introduced in Section III-B can not be directly applied to solve Problem 2 since the forward variable in (2) computes the sequence of *marginally* most probable states, while we are looking for the *jointly* most probable sequence of states given $\boldsymbol{\xi}_1$ and $\boldsymbol{\xi}_T$. As a result, when using (2) there is no guarantee that the returned sequence $\boldsymbol{s}_T^\star$ matches both the spatio-temporal patterns of the demonstrations and the initial/final observations. In terms of the example in Fig. 3, it may return the lower branch as the most likely sequence (due to e.g., more demonstrations), i.e., "pick from the side", even if the desired final configuration as the final observation is that the end-effector is on the top of object.

To overcome this issue, we rely on a modification of the Viterbi algorithm [31]. The classical Viterbi algorithm has been extensively used to find the most likely sequence of states (also called the Viterbi path) in classical HMMs that result in a given stream of observed events. Our method differs from it in two main aspects: **(a)** it works on HSMM instead of HMM; and more importantly **(b)** most observations are missing except those at the first and the last time instants. More specifically, in the absence of intermediate observations the Viterbi algorithm becomes

$$\begin{aligned}\delta_t(j) &= \max_{d\in\mathcal{D}} \max_{i\neq j} \delta_{t-d}(i) a_{ij} p_j(d) \prod_{t'=t-d+1}^{t} \tilde{b}_j(\boldsymbol{\xi}_{t'}), \\ \delta_1(j) &= b_j(\boldsymbol{\xi}_1)\pi_j p_j(1),\end{aligned} \tag{3}$$

where $p_j(d) = \mathcal{N}(d|\mu_j^D, \sigma_j^D)$ is the duration probability of state j, $\delta_t(j)$ is the likelihood of the system being in state j

at time t *and* not in state j at $t+1$, see [31] for details; and

$$\tilde{b}_j(\boldsymbol{\xi}_{t'}) = \begin{cases} \mathcal{N}(\boldsymbol{\xi}_{t'} | \hat{\boldsymbol{\mu}}_j, \hat{\boldsymbol{\Sigma}}_j), & t = 1 \vee t = T; \\ 1, & 1 < t < T. \end{cases}$$

where $(\hat{\boldsymbol{\mu}}_j, \hat{\boldsymbol{\Sigma}}_j)$ is the global Gaussian component j in $\boldsymbol{\Theta}_{\mathsf{a}}$ from (1) given $\boldsymbol{\xi}_{t'}$. Namely, at each time t and for each state j, the two arguments that maximize equation $\delta_t(j)$ are recorded, and a simple backtracking procedure is used to find the most likely state sequence $\boldsymbol{s}_T^\star$. In other words, the above algorithm derives the most-likely sequence $\boldsymbol{s}_T^\star$ for skill a that yields the final observation $\boldsymbol{\xi}_T$, starting from $\boldsymbol{\xi}_1$.

2) ***Trajectory tracking on Manifolds:*** Given $\boldsymbol{s}_T^\star$, we rely on linear quadratic tracking (LQT) to retrieve the optimal reference trajectory. We use a linear double integrator dynamics in the control formulation such that the end-effector follows a virtual spring-damper system. This allows us to compute an optimal and smooth reference trajectory in closed form without knowledge of the actual robot dynamics. As mentioned, the robot state $\boldsymbol{x}_{\mathsf{e}}$ lies in the Riemannian manifold $\mathcal{M}_R = \mathbb{R}^3 \times \mathcal{S}^3 \times \mathbb{R}^1$, which is prohibitive to define the required linear dynamics. However, as in [11], we can exploit the linear tangent spaces to achieve a similar result. We assume the linear double-integrator dynamics is defined in the tangent space of $\boldsymbol{x}$, namely $\mathcal{T}_{\boldsymbol{x}}\mathcal{M}_R$. Also, we define the tangent space robot state as $\mathfrak{x}_t = [\mathrm{Log}_{\boldsymbol{x}_t}(\boldsymbol{x}_t)^\top, \boldsymbol{v}_t^\top]^\top \in \mathcal{T}_{\boldsymbol{x}_t}\mathcal{M}_R$, with velocity $\boldsymbol{v}_t$ and $\boldsymbol{v}_1 = \boldsymbol{0}$. Due to the differential formulation and the definition of the Log-map the robot state in its own tangent space becomes $\mathfrak{x}_t = [\boldsymbol{0}^\top, \boldsymbol{v}_t^\top]^\top$. Then, the optimal control problem becomes

$$\boldsymbol{u}^\star = \arg\min_{\boldsymbol{u}} \sum_{t=1}^{T} \left(\mathrm{Log}_{\boldsymbol{x}_t}(\hat{\boldsymbol{\mu}}_{k_t})^\top \hat{\boldsymbol{\Sigma}}_{k_t}^{-1} \mathrm{Log}_{\boldsymbol{x}_t}(\hat{\boldsymbol{\mu}}_{k_t}) + \boldsymbol{u}_t^\top \boldsymbol{R} \boldsymbol{u}_t \right),$$

$$\text{s.t.} \quad \mathfrak{x}_{t+1} = \boldsymbol{A}\mathfrak{x}_t + \boldsymbol{B}\boldsymbol{u}_t, \quad \mathfrak{x}_{t+1}, \mathfrak{x}_t, \boldsymbol{u}_t \in \mathcal{T}_{\boldsymbol{x}_t}\mathcal{M}_R;$$
$$\boldsymbol{x}_{t+1} = \mathrm{Exp}_{\boldsymbol{x}_t}(\mathfrak{x}_{t+1}) \in \mathcal{M}_R, \quad \boldsymbol{x}_1 = \boldsymbol{x}_{\mathsf{e}}$$
$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{I}\Delta t \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{I}\Delta t \end{bmatrix}$$

where k_t is the t-th component in $\boldsymbol{s}_T^\star$. Specifically, the state error between the desired reference $\hat{\boldsymbol{\mu}}_{k_t}$ and current robot state $\boldsymbol{x}_t$ is computed using the logarithmic map $\mathrm{Log}_{\boldsymbol{x}_t}(\hat{\boldsymbol{\mu}}_{k_t})$ that projects the minimum length path between $\hat{\boldsymbol{\mu}}_{k_t}$ and $\boldsymbol{x}_t$ into $\mathcal{T}_{\boldsymbol{x}_t}\mathcal{M}_R$. We assume $\boldsymbol{0}$ reference velocity, therefore we omit this component in the cost funciton. The covariance matrices $\hat{\boldsymbol{\Sigma}}_{k_t}$ describe the variance and correlation of the robot state variables in a tangent space $\mathcal{T}_{\hat{\boldsymbol{\mu}}_{k_t}}\mathcal{M}_R$. Such covariance matrices must be rotated via parallel transport on the manifold to avoid distortion (see Sec. III-C). Similarly, when we propagate the velocity between consecutive states we have to parallel transport them. For the considered manifold $\mathcal{M}_R$, this can be computed in closed form [11].

The derivation of the optimal control $\boldsymbol{u}^\star$ and state $\boldsymbol{x}^\star$ trajectory follows the ideas of classical optimal control. Similarly to standard LQT, we derive a recursive computation of a feedback and feedforward controller to satisfy the Bellman equation. As the control signal and the controller gains are defined in the tangent space of the current robot state, we

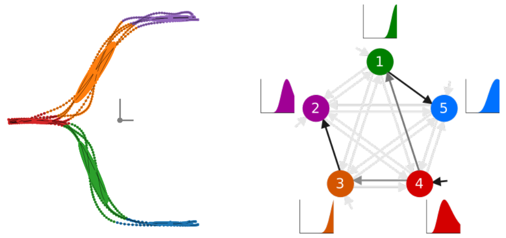

Fig. 3: **Left:** Learned 5-states HSMM in the global frame for skill "pick" in 2-D, where demonstrations are labeled in color by the associated states. The model has a single initial state (number 4 in the right graph) from which two branches (states 1-5 and states 3-2) encode the different instances of the skill. **Right:** transition and duration functions of the HSMM. An arrow's color intensity is proportional to the learned transition probability, where black and light gray respectively depict high and low probabilities.

have to ensure that each variable is parallel transported to this space during recursion. Overall, the Riemannian extension of LQT requires only a minor computational overhead.

B. Skill Sequence Reproduction

The above method would suffice for reproducing a *single* skill, which however can not be applied directly to solve Problem 1. This is because only the final observation after the whole sequence $\mathbf{a}^\star$ is given, while the intermediate observations after each skill are lacking. To overcome this, we propose a two-step solution: **(1)** cascade the models of each skill within $\mathbf{a}^\star$ into one *complete* model $\widehat{\boldsymbol{\Theta}}_{\mathbf{a}^\star}$; **(2)** find the *complete* state sequence $\widehat{s}^\star$ within $\widehat{\boldsymbol{\Theta}}_{\mathbf{a}^\star}$ to reach the goal state with highest probability.

1) ***Cascade multiple HSMMs:*** We first focus our description on cascading two HSMMs, which can then be applied recursively for longer sequences. Consider two TP-HSMMs $\boldsymbol{\Theta}_{\mathsf{a}_1}$ and $\boldsymbol{\Theta}_{\mathsf{a}_2}$, the algorithm for cascading them into $\widehat{\boldsymbol{\Theta}}$ is summarized in Alg. 1. The key insight is that the same model $\boldsymbol{\Theta}_{\mathsf{a}_2}$ is updated *differently* depending on the final component, or terminal state of $\boldsymbol{\Theta}_{\mathsf{a}_1}$ to which $\boldsymbol{\Theta}_{\mathsf{a}_2}$ is cascaded to. This is because each final component encodes different transformations of the task parameters of $\boldsymbol{\Theta}_{\mathsf{a}_1}$ after executing a_1, which in turn results in different ways to update the components in $\boldsymbol{\Theta}_{\mathsf{a}_2}$. Consequently, the composed model $\widehat{\boldsymbol{\Theta}}$ has size $K_1 + K_{1,f} \cdot K_2$, where K_1 and K_2 are the number of components of $\boldsymbol{\Theta}_{\mathsf{a}_1}$ and $\boldsymbol{\Theta}_{\mathsf{a}_2}$, respectively, while $K_{1,f}$ is the number of final components in $\boldsymbol{\Theta}_{\mathsf{a}_1}$. More specifically, Alg. 1 consists of two main steps: **(a)** compute the transition probability from *each* final component in $\boldsymbol{\Theta}_{\mathsf{a}_1}$ to *each* initial component in $\boldsymbol{\Theta}_{\mathsf{a}_2}$; **(b)** modify all components of $\boldsymbol{\Theta}_{\mathsf{a}_2}$ for each *final* component in $\boldsymbol{\Theta}_{\mathsf{a}_1}$ that $\boldsymbol{\Theta}_{\mathsf{a}_2}$ is cascaded to.

To begin with, we recall the precondition and effect model proposed in our earlier work [32]. In particular, the learned precondition model, denoted by $\boldsymbol{\Gamma}_{1,\mathsf{a}}$, contains TP-GMMs for the initial robot state, i.e., $\boldsymbol{\Gamma}_{1,\mathsf{a}} = \{(\hat{\boldsymbol{\mu}}_1^{(p)}, \hat{\boldsymbol{\Sigma}}_1^{(p)}), \forall p \in P_{1,\mathsf{a}}\}$, where $P_{1,\mathsf{a}}$ is the chosen set of task parameters, derived from the initial system state (e.g., initial pose of objects of interest). In addition, we introduce here the final condition model $\boldsymbol{\Gamma}_{T,\mathsf{a}}$, which is learned in a similar way as $\boldsymbol{\Gamma}_{1,\mathsf{a}}$, but for the final robot state, i.e., $\boldsymbol{\Gamma}_{T,\mathsf{a}} = \{(\hat{\boldsymbol{\mu}}_T^{(p)}, \hat{\boldsymbol{\Sigma}}_T^{(p)}), \forall p \in P_{T,\mathsf{a}}\}$, where $P_{T,\mathsf{a}}$ is the chosen set of frames, derived from the final system state. Simply speaking, $\boldsymbol{\Gamma}_{1,\mathsf{a}}$ models the initial

Algorithm 1: Cascading a pair of TP-HSMMs

```
  Input: $(\Theta_{a_1}, \Gamma_{a_1})$ and $(\Theta_{a_2}, \Gamma_{a_2})$.
  Output: $(\widehat{\Theta}, \widehat{\Gamma})$
1 forall final component $k_f \in \Theta_{a_1}$ do
2     Create copy of $\Theta_{a_2}$ as $\Theta_{a_2}^{k_f}$.
3     Compute $\{a_{k_f,k_i}\}$ for all initial $k_i \in \Theta_{a_2}^{k_f}$ by (4).
4     Update $\Theta_{a_2}^{k_f}$ and $\Gamma_{1T,a_2}^{k_f}$ by (5).
5     Cascade $\Theta_{a_1}$ and $\Theta_{a_2}^{k_f}$. Add to $\widehat{\Theta}$.
6 Set additional parameters of $\widehat{\Theta}$.
7 $\widehat{\Gamma} = \{\widehat{\Gamma}_1, \widehat{\Gamma}_T, \widehat{\Gamma}_{1T}\} = \{\Gamma_{1,a_1}, \Gamma_{T,a_2}, \{\Gamma_{1T,a_2}^{k_f}, \forall k_f\}\}$.
```

configuration before executing skill a, while $\Gamma_{T,a}$ models the final configuration afterwards. Furthermore, the learned effect model $\Gamma_{1T,a}$, contains TP-GMMs for the *predicted* final system state, i.e., $\Gamma_{1T,a} = \{\{(\hat{\mu}_{1,o}^{(p)}, \hat{\Sigma}_{1,o}^{(p)}), \forall p \in P_{1,a}\}, \forall o \in O_a \cup e\}$, where $P_{1,a}$ is defined in $\Gamma_{1,a}$. Notice the differences among these three models: the task parameters for $\Gamma_{T,a}$ are computed from the final system state, while those for $\Gamma_{1,a}$ and $\Gamma_{1T,a}$ are extracted from the initial system state. Their derivation is omitted here and we refer the readers to [32]. For the sake of notation, we define $\Gamma_a \triangleq \{\Gamma_{1,a}, \Gamma_{T,a}, \Gamma_{1T,a}\}$.

Then, the transition probability from one final component k_f of Θ_{a_1} to one initial component k_i of Θ_{a_2} is:

$$a_{k_f,k_i} \propto \exp\Big(-\sum_{p \in P_c} KL\big(\Gamma_{T,a_1}^{(p)}(k_f) || \Gamma_{1,a_2}^{(p)}(k_i)\big)\Big), \quad (4)$$

where $KL(\cdot||\cdot)$ is the KL-divergence from [33], $\Gamma_{T,a_1}^{(p)}(k_f)$ is the GMM associated with component k_f for frame p, $\Gamma_{1,a_2}^{(p)}(k_i)$ is the GMM associated with component k_i for frame p; $P_c = P_{T,a_1} \cap P_{1,a_2}$ is the set of *common* frames shared by these two models, which can be forced to be nonempty by always adding the global frame. This process is repeated for all pairs of final components in Θ_{a_1} and initial components in Θ_{a_2}. Note that the out-going probability of any final component in Θ_{a_1} should be normalized.

Secondly, given one final component k_f of Θ_{a_1}, each component k of Θ_{a_2} should be affine-transformed as follows:

$$(\hat{\mu}_k^{(\hat{p})}, \hat{\Sigma}_k^{(\hat{p})}) \triangleq (\mu_k^{(p)}, \Sigma_k^{(p)}) \otimes (b_{k_f}^{(\hat{p})}, A_{k_f}^{(\hat{p})}), \quad (5)$$

where the operation $\otimes$ is defined as the same operation of (1); $(b_{k_f}^{(\hat{p})}, A_{k_f}^{(\hat{p})})$ is the task parameter computed from the *mean* of $\Gamma_{1T,a_1}^{(\hat{p}),o}(k_f)$, where o is the object associated with the old frame p in Θ_{a_1} and $\hat{p}$ is the new frame in $\Gamma_{1T,a_1}^{o}(k_f)$. Note that the change of frames is essential to compute directly all components of Θ_{a_2} given an initial system state of Θ_{a_1}. The same process is also applied to each component of Γ_{1T,a_2} by changing its frames based on $\Gamma_{1T,a_1}^{o}(k_f)$.

Lastly, other model parameters of $\widehat{\Theta}$ such as duration probabilities, initial and final distributions are set with minor changes from Θ_{a_1} and Θ_{a_2}. For instance, the duration probability of Θ_{a_2} is duplicated to k_f multiple copies; the initial distributions Θ_{a_2} are set to zero as the initial states of $\widehat{\Theta}$ correspond to those of the first model Θ_{a_1}; the final components of Θ_{a_1} are removed since the final states of $\widehat{\Theta}$ are now those of Θ_{a_2} updated to its multiple instances.

Algorithm 2: Execute skill sequence $\mathbf{a}^\star$

```
  Input: A, $\mathbf{a}^\star$, $x_1$, $x_G$, and $\{(\Theta_a, \Gamma_a), \forall a \in \mathbf{a}^\star\}$.
1 Compute the composed model $\widehat{\Theta}_{\mathbf{a}^\star}$ using Alg. 1.
2 Compute optimal sequence $\hat{s}^\star$ over $\widehat{\Theta}_{\mathbf{a}^\star}$, given $\xi_1$
    and $\xi_T$ via (3).
3 for each $a_h \in \mathbf{a}^\star$ do    // On-line Execution
4     Observe the current system state $x_{h,1}$.
5     Update the global components in $\hat{s}_h^\star$ given $x_{h,1}$.
6     Track $\hat{s}_h^\star$ by motion control till the end.
```

Now consider the desired skill sequence given as $\mathbf{a}^\star = a_1 a_2 a_3 \cdots a_N$. First, Alg. 1 is applied to $(\Theta_{a_1}, \Gamma_{a_1})$ and $(\Theta_{a_2}, \Gamma_{a_2})$, yielding $(\widehat{\Theta}, \widehat{\Gamma})$. Then, Alg. 1 is applied to $(\widehat{\Theta}, \widehat{\Gamma})$ and $(\Theta_{a_3}, \Gamma_{a_3})$. This process repeats itself until a_N as the end of $\mathbf{a}^\star$, yielding $(\widehat{\Theta}_{\mathbf{a}^\star}, \widehat{\Gamma}_{\mathbf{a}^\star})$. Note that $\widehat{\Theta}_{\mathbf{a}^\star}$ is the *complete* cascaded model in the standard TP-HSMM format.

*2) **State sequence generation and tracking:*** The derived model $\widehat{\Theta}_{\mathbf{a}^\star}$ is used to reproduce the skill sequence $\mathbf{a}^\star$ as follows. Firstly, the initial system state x_1 is obtained (e.g., from perception) and mapped to the initial observation ξ_1. The goal system state x_G, e.g., from a high-level planner, is also mapped to the final observation ξ_T. The total length T is then set to, e.g., the accumulative average length of all skills in $\mathbf{a}^\star$. Then, the same solution for Problem 2 is used to find the most-likely sequence $\hat{s}^\star$ within model $\widehat{\Theta}_{\mathbf{a}^\star}$, given ξ_1 and ξ_T. Note that $\hat{s}^\star$ contains the sub-sequence of components, denoted by $\hat{s}_h^\star$, to be followed for *each* skill $a_h \in \mathbf{a}^\star$. During the task execution of each skill $a_h \in \mathbf{a}^\star$, firstly the current system state $x_{h,0}$ is observed and used to compute the task parameters and update the global GMMs associated with the components in $\hat{s}_h^\star$. Afterwards, the trajectory tracking control described in Sec. V-A.2 is used to track $\hat{s}_h^\star$. This process repeats for the subsequent skills until the end of $\mathbf{a}^\star$, as summarized in Alg. 2. Note that simply tracking $\hat{s}^\star$ without observing the actual intermediate system state $x_{h,1}$ would often fail, due to the perception error and motion noise.

VI. Experiments

We here describe the experiment setup on a 7-DoF robotic manipulator. We consider various assembly tasks consisting of different sequences of skills and show how our framework is used to accomplish such tasks from various initial states. The pipeline is implemented in C++, Python and ROS.

A. Workspace Setup and Manipulation Tasks

The Franka Emika Panda robot (denoted by $\mathfrak{r}$) has 7 DoF and is equipped with a two-fingers gripper, as shown in Fig. 4. The workspace consists of a feeding and inspection platform, where pieces are off-loaded and inspected; and an assembly station where various pieces are assembled into a product. The platform is monitored by a Zivid 3D camera, from which the collected point-clouds are inputs to the point-pair-feature detection algorithm [34]. It provides a 6D pose estimation with around 1 cm accuracy w.r.t the global frame

A	Skill Name	M_{a}	K_{a}	B_{a}	TP_{a}	$t\,(\mathbf{\Theta}_{\mathsf{a}}\,\|\,\mathbf{\Gamma}_{\mathsf{a}})$ [s]
$\mathsf{a}_{\mathtt{gr}}$	`grasp`	35	28	4	$\{\mathfrak{c}\}$	98 | 18
$\mathsf{a}_{\mathtt{ro}}$	`re_orient`	18	12	2	$\{\mathfrak{c}, \mathfrak{r}, \mathfrak{g}\}$	54 | 12
$\mathsf{a}_{\mathtt{tl}}$	`translate`	9	8	1	$\{\mathfrak{c}, \mathfrak{r}, \mathfrak{g}\}$	16 | 4
$\mathsf{a}_{\mathtt{at}}$	`attach`	7	6	1	$\{\mathfrak{c}, \mathfrak{g}\}$	12 | 4
$\mathsf{a}_{\mathtt{dp}}$	`drop`	31	18	3	$\{\mathfrak{c}, \mathfrak{r}, \mathfrak{g}\}$	120 | 18
$\mathsf{a}_{\mathtt{pk}}$	`pick`	6	5	1	$\{\mathfrak{p}, \mathfrak{r}\}$	9 | 3
$\mathsf{a}_{\mathtt{is}}$	`insert`	5	4	1	$\{\mathfrak{p}, \mathfrak{r}\}$	10 | 5

TABLE I: Demonstrated skills A, number of demonstrations M_{a}, number of components K_{a}, number of branches B_{a}, choice of task parameters TP_{a}, and training time for $\mathbf{\Theta}_{\mathsf{a}}$ and $\mathbf{\Gamma}_{\mathsf{a}}$.

(denoted by $\mathfrak{g}$). A task-space incomplete impedance controller [35] is used to track Cartesian reference trajectories.

We consider parts of an E-bike motor assembly process in our tasks. **Task-1**: pick a non-defective metal cap from the platform, and attach it to the top of a metal peg on the assembly station; **Task-2**: pick a defective metal cap from the platform, and drop it to a container. Both the cap $\mathfrak{c}$ and peg $\mathfrak{p}$ are components of the e-bike motor. During kinesthetic teaching, the state of the end-effector is fetched directly from the on-board control manager. Demonstrations are recorded at $50\,\mathrm{Hz}$, while the task-space impedance controller runs at $1\,\mathrm{kHz}$. As summarized in Table. I, we demonstrated in total 7 skills relevant to the task: `grasp` where the robot grasps $\mathfrak{c}$ in four different cases: via the top, the side, flat-right and flat-left (denoted by $\mathsf{a}_{\mathtt{gr}}$); `re-orient` where the robot re-orients $\mathfrak{c}$ from lying flat-left or flat-right to stand-up (by $\mathsf{a}_{\mathtt{ro}}$); `translate` where the robot moves $\mathfrak{c}$ to the edge of the platform (by $\mathsf{a}_{\mathtt{tl}}$); `attach` where the robot attaches $\mathfrak{c}$ on the top of $\mathfrak{p}$ (by $\mathsf{a}_{\mathtt{at}}$); `drop` where the robot drops $\mathfrak{c}$ into the container facing inwards or outwards (denoted by $\mathsf{a}_{\mathtt{dp}}$); `pick` where the robot picks $\mathfrak{p}$ (denoted by $\mathsf{a}_{\mathtt{pk}}$); `insert` where the robot inserts $\mathfrak{p}$ (denoted by $\mathsf{a}_{\mathtt{is}}$). With these skills, the two tasks above can be re-stated as follows: **Task-1**: If $\mathfrak{c}$ is standing-up, the skill sequence is given by $\mathbf{a}_1 = \mathsf{a}_{\mathtt{pk}}\mathsf{a}_{\mathtt{is}}\mathsf{a}_{\mathtt{gr}}\mathsf{a}_{\mathtt{at}}$; If $\mathfrak{c}$ is lying-flat, the skill sequence is given by $\mathbf{a}_1' = \mathsf{a}_{\mathtt{pk}}\mathsf{a}_{\mathtt{is}}\mathsf{a}_{\mathtt{gr}}\mathsf{a}_{\mathtt{ro}}\mathsf{a}_{\mathtt{gr}}\mathsf{a}_{\mathtt{at}}$. **Task-2**: If $\mathfrak{c}$ is standing-up, the skill sequence is given by $\mathbf{a}_2 = \mathsf{a}_{\mathtt{gr}}\mathsf{a}_{\mathtt{tl}}\mathsf{a}_{\mathtt{gr}}\mathsf{a}_{\mathtt{dp}}$; If $\mathfrak{c}$ is lying-flat, the skill sequence is given by $\mathbf{a}_2' = \mathsf{a}_{\mathtt{gr}}\mathsf{a}_{\mathtt{dp}}$;

Clearly, the desired skill sequence for different tasks depends on the object state. For **Task-1**, since the skill `attach` is only taught when the cap is grasped from the top (i.e., not from the side), a cap initially lying flat on the platform needs to be first re-oriented to a stand-up state. On the contrary, for **Task-2**, since the skill `drop` is only allowed when the cap is grasped from the side (i.e., not from the top), a cap initially standing on the platform needs to be first translated to the edge of the platform, and then grasped from the side. The re-orientation and grasping from the side are only taught at the edge of the platform to avoid collisions.

A video of the **Task-2** experiments is attached as supplementary material, and a extended version including **Task-1** results can be found in https://youtu.be/dRGLadt32o4.

B. Results

Following Alg. 2, both the TP-HSMM model $\mathbf{\Theta}_{\mathsf{a}}$ and the condition model $\mathbf{\Gamma}_{\mathsf{a}}$ are learned for each skill described above. The total number of components, number of branches, task parameters and training times are listed in Table I. Note that skills with several branches have more components and in general take longer to train. Thus, such models should be learned off-line before online execution. Single skills can be easily reproduced given various initial and final observation by following Sec. V-A. Table II summarizes the trajectory length and computation time for the optimal state sequence and optimal control. It shows that the modified Viterbi algorithm (3) and the optimal tracking control are fast enough to be used for online execution. Reproduction of a single skill has almost 100% success rate across various initial configurations that are similar to demonstrated ones. For instance, the `Pick` and `Insert` skills are retrieved when the cap is placed at different locations on the platform surface. In general, when the initial configuration differs drastically from any demonstration, the performance degrades while the resulting trajectory stills resembles similar movement.

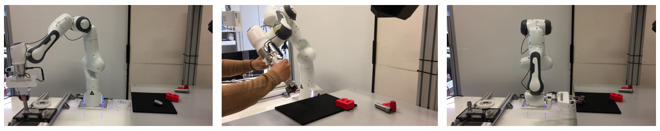
Fig. 4: The experiment setup (*left*), and snapshots of kinesthetic teaching and execution of skills $\mathsf{a}_{\mathtt{pc}}$ and $\mathsf{a}_{\mathtt{ro}}$ (*middle-right*).

Skill	T	$t\,(\boldsymbol{s}^\star\,\|\,\boldsymbol{u}^\star)$[s]	Skill	T	$t\,(\boldsymbol{s}^\star\,\|\,\boldsymbol{u}^\star)$[s]
$\mathsf{a}_{\mathtt{gr}}$	105	1.5 | 0.2	$\mathsf{a}_{\mathtt{ro}}$	175	0.6 | 0.3
$\mathsf{a}_{\mathtt{tl}}$	90	0.1 | 0.1	$\mathsf{a}_{\mathtt{at}}$	135	0.5 | 0.3
$\mathsf{a}_{\mathtt{dp}}$	132	0.9 | 0.2	$\mathsf{a}_{\mathtt{pk}}$	116	0.2 | 0.2

TABLE II: The trajectory length of T time steps and the average computation time of $\boldsymbol{s}^\star$ and $\boldsymbol{u}^\star$ for reproducing each single skill.

For each skill sequence described for **Task-1** and **Task-2**, we follow Sec. V-B to reproduce them under various initial system states. Since the sub-sequence $\mathsf{a}_{\mathtt{pk}}\mathsf{a}_{\mathtt{dp}}$ manipulates only the peg $\mathfrak{p}$ and both skills have only one branch, we treat this sequence independently from other sub-sequences which manipulate only the cap $\mathfrak{c}$. Table III summarizes for each sub-sequence the size of the corresponding composed model $\widehat{\mathbf{\Theta}}$, the number of its initial and final components, and the time taken to compute $\widehat{\mathbf{\Theta}}$ and $\boldsymbol{s}^\star$, respectively. As discussed in Sec. V-B.1, the complexity of $\widehat{\mathbf{\Theta}}$ grows combinatorially with the number of final states in each skill model. For long sequences such as $\mathbf{a}_1'$ and $\mathbf{a}_2'$, it takes much longer to compute $\widehat{\mathbf{\Theta}}$ than short sequences such as $\mathbf{a}_1$ and $\mathbf{a}_2$. Note that once such models are computed, they can be saved as a standard TP-HSMM model. During execution, the initial system state is obtained, e.g., from the perception system while the goal state is specified directly. Then the model $\widehat{\mathbf{\Theta}}$ is loaded directly without the need for recalling Alg. 1. It takes on average 0.5s to compute the complete sequence $\boldsymbol{s}^\star$ given $\widehat{\mathbf{\Theta}}$ for tasks with two simple skills, and around 17s for tasks with four complex skills. As stated in Alg. 2, each skill $a_h \in \mathbf{a}^\star$ is executed by tracking the sub-sequence $\boldsymbol{s}_h^\star$ given the current system state. Supplementary videos show the reproduction of each task under various system states.

C. Discussion

As shown in Table (III), the computation time grows combinatorially to the number of *final* states for each skill

$\mathsf{a}^\star$	K	E	$K_i \mid K_f$	T	$t\,(\widehat{\boldsymbol{\Theta}} \mid \boldsymbol{s}^\star)$[s]
$\mathsf{a}_{\texttt{pk}}\mathsf{a}_{\texttt{is}}$	11	10	1 | 1	216	11 | 0.1
$\mathsf{a}_{\texttt{gr}}\mathsf{a}_{\texttt{dp}}$	100	92	4 | 12	214	21 | 1
$\mathsf{a}_{\texttt{gr}}\mathsf{a}_{\texttt{tl}}\mathsf{a}_{\texttt{gr}}\mathsf{a}_{\texttt{dp}}$	460	440	4 | 76	435	57 | 19
$\mathsf{a}_{\texttt{gr}}\mathsf{a}_{\texttt{at}}$	52	48	4 | 4	239	10 | 0.4
$\mathsf{a}_{\texttt{gr}}\mathsf{a}_{\texttt{ro}}\mathsf{a}_{\texttt{gr}}\mathsf{a}_{\texttt{at}}$	492	487	4 | 80	521	76 | 17

TABLE III: The number of components K, edges E, initial K_i and final states K_f of the model $\widehat{\boldsymbol{\Theta}}_{\mathsf{a}}$. The trajectory length of T time steps, the total computation time for $\widehat{\boldsymbol{\Theta}}_{\mathsf{a}}$ and the associated $\boldsymbol{s}^\star$.

in the sequence. So, the more branches there are in one skill, the more time consuming to compose it with other skills. This is because all branches are compared in terms of likelihood for both the current skill and the subsequent one. Thus our framework, at its present state cannot be used for real-time replanning. Nonetheless, several methods can be used to accelerate this process: **(1)** the skill sequence can often be decomposed in independent sub-sequences, which can be planned separately; **(2)** prune early-on transitions whose probability is below a threshold both in the skill model and the transition probability from (4); **(3)** construct $\widehat{\boldsymbol{\Theta}}_{\mathsf{a}}$ *specifically* for the given initial and final states, i.e., combine (4) with predicted observation probability of k_i. Also, certain skills may be formulated with *free* task parameters, which can be optimized in a general way for various tasks. This issue is not addressed here since all task parameters are assumed to be attached to physical objects. Combination of these methods will be addressed in future research.

VII. Conclusion

We presented a framework for learning and sequencing of robot manipulation skills that are object-centric and learned from demonstration. In our framework we learn generic skills that encapsulate different instantiations, whose activation rely on the initial and goal state of the system. Given a goal configuration and a manipulation task as skill sequence, our framework computes the optimal state sequence to accomplish the task. For successful execution, we also proposed a Riemannian optimal controller in order to calculate a smooth reference trajectory in the manifold of end-effector poses. In contrast to previous works, we experimentally studied our framework on real complex assembly tasks, showing that our approach scales to long sequences of manipulation skills.

References

[1] S. Bedaf, P. Marti, F. Amirabdollahian, and L. de Witte, "A multi-perspective evaluation of a service robot for seniors: the voice of different stakeholders," *Disabil Rehabil*, vol. 13:6, pp. 592–599, 2018.

[2] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[3] S. M. LaValle, *Planning Algorithms*. Cambridge univ. press, 2006.

[4] D. Murray and J. J. Little, "Using real-time stereo vision for mobile robot navigation," *AURO*, vol. 8, no. 2, pp. 161–171, 2000.

[5] M. Goodrich and A. Schultz, "Human-robot interaction: A survey," *Found. Trends Hum.-Comput. Interact.*, vol. 1:3, pp. 203–275, 2007.

[6] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Probabilistic movement primitives," in *NeurIPS*, 2013, pp. 2616–2624.

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

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

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

[10] M. Y. Seker, M. Imre, J. H. Piater, and E. Ugur, "Conditional neural movement primitives," in *R:SS*, 2019.

[11] M. Zeestraten, "Programming by demonstration on Riemannian manifolds," 2018, PhD thesis.

[12] N. Jaquier, L. Rozo, S. Calinon, and M. Bürger, "Bayesian optimization meets Riemannian manifolds in robot learning," in *CoRL*.

[13] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*. Elsevier, 2004.

[14] M. Fox and D. Long, "PDDL2.1: An extension to PDDL for expressing temporal planning domains," *Journal of Artificial Intelligence Research*, vol. 20, pp. 61–124, 2003.

[15] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters, "An algorithmic perspective on imitation learning," *Foundations and Trends in Robotics*, vol. 7, no. 1-2, pp. 1–179, 2018.

[16] S. Niekum, S. Osentoski, G. Konidaris, S. Chitta, B. Marthi, and A. G. Barto, "Learning grounded finite-state representations from unstructured demonstrations," *IJRR*, vol. 34, no. 2, pp. 131–157, 2015.

[17] L. Rozo, D. Bruno, S. Calinon, and D. G. Caldwell, "Learning optimal controllers in human-robot cooperative transportation tasks with position and force constraints," in *IEEE/RSJ IROS*, 2015, pp. 1024–1030.

[18] G. Ye and R. Alterovitz, "Guided motion planning," in *Robotics research*. Springer, 2017, pp. 291–307.

[19] J. Silvério, L. Rozo, S. Calinon, and D. G. Caldwell, "Learning bimanual end-effector poses from demonstrations using task-parameterized dynamical systems," in *IEEE/RSJ IROS*, 2015, pp. 464–470.

[20] S. Manschitz, J. Kober, M. Gienger, and J. Peters, "Learning to sequence movement primitives from demonstrations," in *IEEE/RSJ IROS*, 2013, pp. 4414–4421.

[21] ——, "Probabilistic progress prediction and sequencing of concurrent movement primitives," in *IEEE/RSJ IROS*, 2015, pp. 449–455.

[22] P. Pastor, M. Kalakrishnan, L. Righetti, and S. Schaal, "Towards associative skill memories," in *IEEE Humanoids*, 2012, pp. 309–315.

[23] K. Gräve and S. Behnke, "Learning sequential tasks interactively from demonstrations and own experience," in *IEEE/RSJ IROS*, 2013, pp. 3237–3243.

[24] F. Stulp, E. A. Theodorou, and S. Schaal, "Reinforcement learning with sequences of motion primitives for robust manipulation," *IEEE T-RO*, vol. 28, no. 6, pp. 1360–1370, 2012.

[25] R. Lioutikov, O. Kroemer, G. Maeda, and J. Peters, "Learning manipulation by sequencing motor primitives with a two-armed robot," in *Intelligent Autonomous Systems 13*. Springer, 2016, pp. 1601–1611.

[26] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto, "Robot learning from demonstration by constructing skill trees," *IJRR*, vol. 31, no. 3, pp. 360–375, 2012.

[27] K. Muelling, J. Kober, and J. Peters, "Learning table tennis with a mixture of motor primitives," in *Humanoids*, 2010, pp. 411–416.

[28] O. Kroemer, C. Daniel, G. Neumann, H. Van Hoof, and J. Peters, "Towards learning hierarchical skills for multi-phase manipulation tasks," in *IEEE ICRA*, 2015, pp. 1503–1510.

[29] A. K. Tanwani and S. Calinon, "Learning Robot Manipulation Tasks with Task-Parameterized Hidden Semi-Markov Model," *IEEE RA-L*, pp. 1–8, 2016.

[30] S.-Z. Yu and H. Kobayashi, "A hidden semi-Markov model with missing data and multiple observation sequences for mobility tracking," *Signal Processing*, vol. 83, no. 2, pp. 235–250, 2003.

[31] G. D. Forney, "The Viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

[32] L. Schwenkel, M. Guo, and M. Bürger, "Optimizing sequences of probabilistic manipulation skills learned from demonstration," in *CoRL*, 2019.

[33] J. R. Hershey and P. A. Olsen, "Approximating the Kullback Leibler divergence between Gaussian mixture models," in *IEEE ICASSP*, vol. 4, 2007, pp. IV–317.

[34] M. Nixon and A. Aguado, *Feature extraction and image processing for computer vision*. Academic Press, 2019.

[35] N. Hogan, "Impedance control: An approach to manipulation: Part II–Implementation," *Journal of dynamic systems, measurement, and control*, vol. 107, no. 1, pp. 8–16, 1985.