Interactive Human-in-the-loop Coordination of Manipulation Skills Learned from Demonstration

Meng Guo[1] and Mathias Bürger[2]

***Abstract*— Learning from demonstration (LfD) provides a fast, intuitive and efficient framework to program robot skills, which has gained growing interest both in research and industrial applications. Most complex manipulation tasks are long-term and involve a set of skill primitives. Thus it is crucial to have a reliable coordination scheme that selects the correct sequence of skill primitive and the correct parameters for each skill, under various scenarios. Instead of relying on a precise simulator, this work proposes a human-in-the-loop coordination framework for LfD skills that: builds parameterized skill models from kinesthetic demonstrations; constructs a geometric task network (GTN) on-the-fly from human instructions; learns a hierarchical control policy incrementally during execution. This framework can reduce significantly the manual design efforts, while improving the adaptability to new scenes. We show on a 7-DoF robotic manipulator that the proposed approach can teach complex industrial tasks such as bin sorting and assembly in less than 30 minutes.**

I. Introduction

Robots have been making their ways out of the closed fences in industrial factories. Collaborative robots (*cobots*) are intended for direct human interactions within a shared space. This workspace is much more dynamic than the precisely-arranged structures, e.g., assembly lines. Moreover, the designated tasks are often low in repetition and high in variance. Namely, the robots should adapt to different tasks and different scenarios. For instance, the same service robot might be used for cleaning, packing and transportation. Two characteristics are essential for such use cases: (i) the ability to *quickly* programme robot for different tasks; (ii) the learned task policy should *adapt* automatically to unforeseen situations. Most motion planners however require precise modeling of the scene and the robot [1], thus difficult to configure for non-technical end-users. Instead, learning from demonstrations (LfD) provides an intuitive and efficient method to program robot skills even for layman.

Moreover, instead of a single motion, complex manipulation tasks often contain multiple branches of skill sequences that share some common skills. The planning process should generate the right sequence of skills and their parameters under different scenarios. For instance, as discussed in later the experiment, the bin-picking task involves to pick the object differently from the box, clear it from the corners if needed, re-orient it to reveal the barcode, and show the barcode to the scanner. To choose the correct skill sequence is essential for flexible robotic systems across various applications. Such transitions among the skills and the associated conditions

[1]College of Engineering, Peking University, China. [2]Bosch Center for Artificial Intelligence (BCAI), Germany. Corresponding author: Meng Guo. meng.guo@pku.edu.cn.

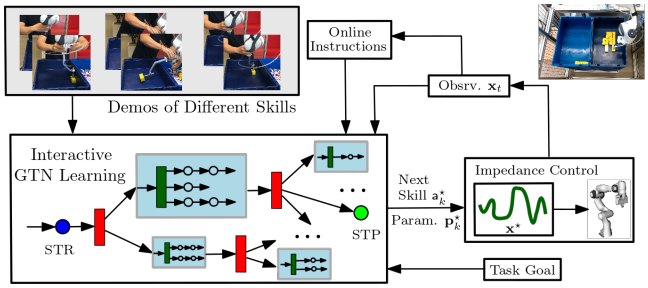

Fig. 1: Diagram of the proposed human-in-the-loop skill coordination scheme. Once the skill primitives are demonstrated offline, the task network and the associated policy are learned online given real-time human instructions and state observations.

are often difficult and tedious to specify manually. In fact, both self-adaptation and autonomous decision-making are important design principles of Industry 4.0 systems [2].

Lastly, besides kinesthetic demonstrations, human operators can also provide *interactive* guidance during online task execution. They are often more efficient and effective than offline simulated data, see [3], [4]. However, the amount of such inputs should not be excessive and only required under appropriate indications, e.g., when the confidence of a decision process is low. In other words, transparency during the interaction is crucial for a human-in-the-loop system.

As shown in Fig. 1, in this work we address these issues by proposing a human-in-the-loop coordination framework for skills learned from demonstrations. In particular, the backbone of the coordination framework is a geometric task network (GTN) which consists of the primitive skills as nodes and their transition relations as edges. Given a manipulation task, the learned task network can decide not only which skill to execute given the current system state, but also the associated parameters. The proposed algorithm first constructs the network structure, then learns the underlying hierarchical selectors based on the geometric constraints among the robots and the objects. Human instructions are only required at the first few executions to improve the task network on-the-fly. Several industrial applications are studied on hardware to validate the performance.

Main contribution of this work is threefold: (i) a more general task-parameterized model for skills with multiple branches; (ii) a novel structure as geometric task networks (GTN) for coordinating LfD skills, which is fully-explainable regarding the underlying decisions; (iii) an online, human-in-the-loop and interactive learning algorithm, which is extremely data-efficient and intuitive regarding the required human instructions.

II. Related Work

A. Learning from Demonstration

Compared with traditional motion planning from [1], learning from demonstration (LfD) is an intuitive way to transfer human skills to robots, see [5], [6], [7]. Various teaching methods can be used such as kinesthetic teaching in [5], tele-operation in [8], and visual demonstration in [7]. Different skill models are proposed to abstract these demonstrations: full trajectory of robot end-effector in [6], dynamic movement primitives (DMPs) in [9], [10], task-parameterized Gaussian mixture models (TP-GMMs) in [5], [11] which extend GMMs by incorporating observations from different perspectives (so called task parameters), task-parametrized hidden semi-Markov models (TP-HSMMs) in [12], [13], [14], and deep neural networks [7]. In this work, we adopt the TP-HSMM representation to extract both temporal and spatial features from few human teachings, while allowing generalization over multiple task parameters.

B. Task and Motion Planning

Task planning focuses on constructing a discrete high-level plan via abstract decision-making (e.g., via logic-reasoning from [15]), while motion planning addresses the low-level sensing and control problem of a dynamic system, see [1]. The area of task and motion planning (TAMP) attempts to improve the synergies between them. The planning process over the state graph consists of searching over both discrete skill sequences and the continuous parameters, see [16], [17], [18]. A conjugate view of the state graph is the so-called skill graph, where instead the nodes are primitive skills and edges are implicit state abstractions, see [19], [20], [21], [22]. The work in [20] extends the hierarchical task networks (HTN) to conjugate task graph (CTG) without any parameterization on the skill primitives. Moreover, [22] calls such graph as maneuver automaton, which however is *manually* designed instead of learned, whereas [19] require similar structural supervision during training. The method in [21] relies on "change point" detection to segment these task demonstrations with simple 2D models, while [23] assumes each skill primitive is parameterized to only one object frame. In this work, we adopt this conjugate perspective, but with an embedded hierarchical structure of selectors. Moreover, both the graph structure and the associated geometric constraints are learned online, without manual specifications.

C. Human-in-the-loop Learning

Human operators can also provide interactive instructions during online task execution, which are often more efficient and effective than offline simulated data, see [3], [4]. Human inputs can be of different formats such as qualitative preferences [4], directly modifying the underlying algorithms [3], additional actions [24]. In this work, the human operators are only queried when the decision confidence is low, and the expected inputs are simple instructions such as the next desired skill or branch name.

III. Preliminaries

A. Multi-nomial Classification

Multi-nomial or multi-class classification is the problem of classifying instances into one of several classes, see [25]. More precisely, consider the training data $\{(\boldsymbol{y}_m, k_m)\}$ where $\boldsymbol{y}_m \in \mathbb{R}^N$ is the feature vector and $k_m \in \{1, \cdots, K\}$ is the set of possible classes. Our goal is to learn a classifier $f : \mathbb{R}^N \to \{1, \cdots, K\}$ that $f(\boldsymbol{y})$ predicts the most likely class of a new point $\boldsymbol{y} \in \mathbb{R}^N$. Various algorithms have been proposed such as support vector machines, logistic regression, k-nearest neighbors, naive Bayes and neural networks, see [25] for details. An extremely data-efficient yet effective method is to use logistic regression along with the *one-vs-rest* strategy, by maximizing the likelihood of correctly classifying all training data. More specifically, the probability of $\boldsymbol{y}_m$ belonging to class k_m is given by

$$p(k_m \,|\, \boldsymbol{y}_m) = \sigma(\boldsymbol{\beta}^\intercal \boldsymbol{y}_m), \tag{1}$$

where $\sigma(t) = 1/(1 + e^{-t})$ is the logistic function for $t \in \mathbb{R}$; $\boldsymbol{\beta} \in \mathbb{R}^N$ is the model parameter as the weight vector.

Logistic regression is used here (instead of SVMs and DNNs) as: (i) it requires much less training data; (ii) it offers probabilistic measures that can be used as confidence in a decision; (iii) the decision model is more interpretable as the weighted combination of the feature variables.

B. Skill Model as TP-HSMM

As proposed in [5], [12], [13], the skill model $\boldsymbol{\theta}_{\mathsf{a}}$ of a primitive skill a as the TP-HSMM is defined as:

$$\boldsymbol{\theta}_{\mathsf{a}} = \left\{ \{a_{hk}\}_{h=1}^{K},\, (\mu_k^D, \sigma_k^D),\, \boldsymbol{\gamma}_k \right\}_{k=1}^{K}, \tag{2}$$

where a_{hk} is the transition probability from component h to component k; (μ_k^D, σ_k^D) describe the duration Gaussian model of component k; and $\boldsymbol{\gamma}_k$ is component k of a TP-GMM: $\boldsymbol{\gamma} = \{\pi_k, \{\boldsymbol{\mu}_k^p, \boldsymbol{\Sigma}_k^p\}_{p=1}^P\}_{k=1}^K$, 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 mean and covariance of the k-th Gaussian component within frame p. Frames provide observations of the *same* data but from different perspectives, the instantiation of which is called a task parameter. An Expectation-Maximization (EM) method from [26] is used to optimize this model.

IV. Problem Description

Consider a multi-DoF robotic arm within a static and known workspace, of which the end-effector has state $\boldsymbol{r}$ such as its 6-D pose and gripper state. Also, there are multiple objects of interest denoted by $\mathsf{O} = \{\mathsf{o}_1, \cdots, \mathsf{o}_J\}$. Each object is described by its state $\boldsymbol{p}_{\mathsf{o}}$ such as its 6-D pose.

Moreover, there is a set of *primitive* skills that enable the robot to manipulate these objects, denoted by $\mathsf{A} = \{\mathsf{a}_1, \mathsf{a}_2, \cdots, \mathsf{a}_H\}$. For each skill, a human user performs several kinesthetic demonstrations on the robot. Particularly, for skill $\mathsf{a} \in \mathsf{A}$, the set of objects involved is given by $\mathsf{O}_{\mathsf{a}} \subseteq \mathsf{O}$ and the set of demonstrations is given by $\boldsymbol{D}_{\mathsf{a}} = \{\mathsf{D}_1, \cdots, \mathsf{D}_{M_{\mathsf{a}}}\}$, where each demonstration D_m is a timed

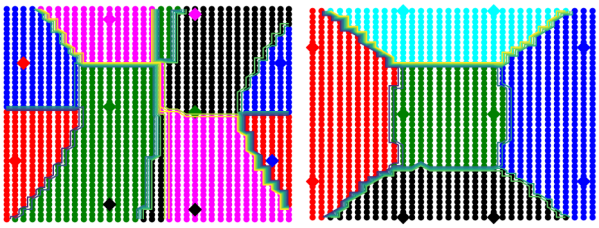

Fig. 2: Comparison between branch selection based on the Gaussian preconditions (left) and the proposed selector (right), for the bin-picking skill with 5 branches (four sides and the center). Note that different colors indicate the predicted branches at that sample point (in dots), while the *projected* training data are indicated as diamonds (left).

sequence of states $\mathbf{s} \in \mathbf{s}$ that consists of the end-effector state $\boldsymbol{r}$ and object states $\{\boldsymbol{p}_{\mathsf{o}}, \mathsf{o} \in \mathsf{O}_{\mathsf{a}}\}$, i.e., $\mathsf{D}_m = \left[\mathbf{s}_t\right]_{t=1}^{T_m} = \left[\left(\boldsymbol{r}_t, \{\boldsymbol{p}_{t,\mathsf{o}}, \mathsf{o} \in \mathsf{O}_{\mathsf{a}}\}\right)\right]_{t=1}^{T_m}$. Via a combination of these skills, the objects can be manipulated to reach different states.

We consider a generic manipulation *task*, which however consists of many instances. Each problem instance is specified by an initial state $\mathbf{s}_0$ and a set of desired goal states $\mathbf{s}_G$. A task is solved when the system state is changed from $\mathbf{s}_0$ to $\mathbf{s}_G$. Then the problem statement is as follows:

Problem 1: Given a new task $(\mathbf{s}_0, \mathbf{s}_G)$, determine (i) the discrete sequence of skills; and (ii) the continuous robot trajectory to execute each skill. ■

We are interested in solving complex manipulation tasks where the sequence of desired skills and the associated trajectories change significantly within different scenarios.

V. Proposed Solution

In this section, we present the main components of the proposed solution: first, we introduce an extension to the current skill model learning; then, we show how to construct a Geometric Task Network (GTN) for a given task; lastly, we describe how both the skill model and the task network can be improved during online execution with human inputs.

A. Primitive Skills Learning

As illustrated in Fig. 1, there are often multiple ways of executing the same skill under different scenarios (called *branches*). For instance, there are five different ways of picking objects from a bin, i.e., approaching with different angles depending on the distances to each boundary. Our earlier work [13] proposed a learning algorithm for TP-HSMM with multiple branches, and moreover a precondition model that chooses the best branch based on the first GMMs of all branches. However, this model requires a large number of demos to cover the area of interest and does not generalizes well to new scenarios. This is mainly due to the usage of Gaussian clustering over few samples in high dimensions. Fig. 2 shows one example of the bin-picking skill.

To overcome this limitation, we propose a *branch selector* as an extension to the original TP-HSMM model $\boldsymbol{\theta}_{\mathsf{a}}$ in Sec. III-B. Consider a skill primitive a with M demonstrations and B different branches. Furthermore, each execution trajectory or demonstration of the skill is denoted by $\mathsf{J}_m = \left[\mathbf{s}_t\right]_{t=1}^{T_m}$, which is associated with *exactly* one branch $b_m \in B_{\mathsf{a}} = \{1, \cdots, B\}$. Denote by $\boldsymbol{J}_{\mathsf{a}}$ the set of such trajectories, initialized to be the set of demos $\boldsymbol{D}_{\mathsf{a}}$. As mentioned in Sec. III-B, the frames associated with J_m are computed from the initial state $\mathbf{s}_0$, by abstracting the coordinates of the robot and the objects, denoted by:

$$(\mathbf{F}_0, \mathbf{F}_1, \cdots, \mathbf{F}_P), \tag{3}$$

where $\mathbf{F}_p = (\mathbf{b}^p, \mathbf{A}^p)$ is the coordinate of frame $p \in \{1, \cdots, P\}$; their order can be freely chosen but fixed afterwards. Then, we can derive the *feature* vector:

$$\mathbf{v}_m = \left(\mathbf{F}_{01}, \mathbf{F}_{12}, \cdots, \mathbf{F}_{(P-1)P}\right), \tag{4}$$

where $\mathbf{F}_{ij} = (\mathbf{b}_{ij}, \boldsymbol{\alpha}_{ij}) \in \mathbb{R}^6$ is the relative transformation from frame $\mathbf{F}_i$ to frame $\mathbf{F}_j$: $\mathbf{b}_{ij} \in \mathbb{R}^3$ is the relative pose and $\boldsymbol{\alpha}_{ij} \in \mathbb{S}^3$ is the relative orientation. Thus, given $\boldsymbol{J}_{\mathsf{a}}$, we can construct the *training data* for the branch selector:

$$\boldsymbol{\tau}_{\mathsf{a}}^{\mathsf{B}} = \{(\mathbf{v}_m, b_m), \forall \mathsf{J}_m \in \boldsymbol{J}_{\mathsf{a}}\}, \tag{5}$$

where b_m is the branch label of trajectory J_m; $\mathbf{v}_m$ is the associated feature vector. Then the branch selector, denoted by $\mathcal{C}_{\mathsf{a}}^{\mathsf{B}}$, can be learned via any multi-nomial classification algorithms. As described in Sec. III-A, logistic regression under the "one-vs-rest" strategy yields an effective selector from few training samples. More comparisons are given in Sec. VI. Afterwards, given a new scenario with state $\mathbf{s}_t$, the probability of choosing branch b is given by:

$$\rho_b = \mathcal{C}_{\mathsf{a}}^{\mathsf{B}}(\mathbf{s}_t, b), \ \forall b \in B_{\mathsf{a}}, \tag{6}$$

where $\rho_b \in [0, 1]$. Since most skills contain two or three frames, the feature vector $\mathbf{v}_m$ in (4) normally has dimension 6 or 12. Fig. 2 illustrates much better classification results compared with the Gaussian preconditions [13].

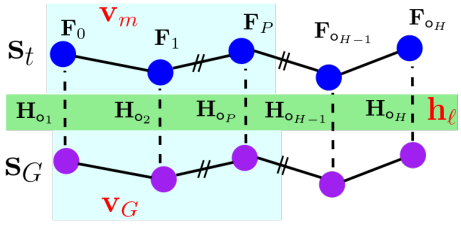

Fig. 3: Illustration of the computation of feature vectors $\mathbf{v}_m$ for the edge selector and $\mathbf{h}_\ell$ for the branch selector, given skill frames $\mathbf{F}_p$.

B. Task Network Construction

As mentioned in Sec. I, complex manipulation tasks often contain various sequences of skills to account for different scenarios. A high-level abstraction of such relations is referred as task networks [20]. A valid plan evolves by transition from one skill to another until the task is solved. Even though the graph structure can be sketched easily, the *conditions* on these transitions are particularly difficult and tedious to specify manually. To overcome this limitation, we propose a novel coordination structure as geometric task networks (GTNs) [27], where the conditions are learned from the task execution results.

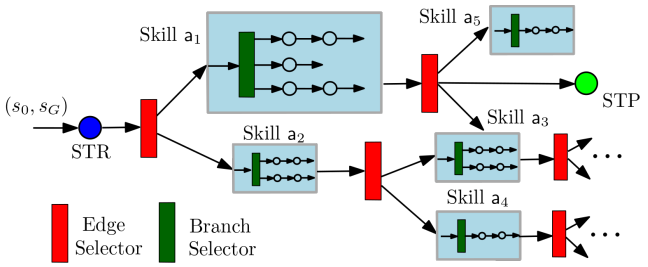

Fig. 4: The proposed network structure over primitive skills, where the conditions for selecting edges and branches are critical for the execution.

1) Network Structure: As illustrated in Fig. 4, a GTN has a simple structure defined by the triple $\mathcal{G} = (V, E, f)$. The set of nodes V is a subset of the primitive skills A; the set of edges $E \subseteq V \times V$ contains the allowed transitions from one skill to another; the function $f : v \to \mathcal{C}$ maps each node to a edge selector w.r.t. all of its outgoing edges. Intuitively, (V, E) specifies how skills can be executed consecutively for the given task, while function $f(v)$ models the different geometric constraints among the objects and the robot, for the outgoing edges of node v. Note that $f(\cdot)$ is *explicitly* conditioned on both the current system state and the goal state. Its detailed model is given in the sequel.

2) Learning from Complete Plans: Without loss of generality, a complete plan of the considered problem in Problem 1 is given by the following sequence:

$$\boldsymbol{\xi} = \underline{\mathsf{a}}\mathbf{s}_0\, \mathsf{a}_0\, \mathbf{s}_1\, \mathsf{a}_1\, \mathbf{s}_2 \cdots \mathbf{s}_G \bar{\mathsf{a}}, \tag{7}$$

where $\underline{\mathsf{a}}$ and $\bar{\mathsf{a}}$ are virtual "start" and "stop" skills, respectively. For different initial and goal states (i.e., problems) of the same task, the resulting plans can be different. Denoted by $\boldsymbol{\Xi} = \{\boldsymbol{\xi}\}$ the set of complete plans for a set of given problems. Then, for each "action-state-action" triple $(\mathsf{a}_n, \mathbf{s}_{n+1}, \mathsf{a}_{n+1})$ within $\boldsymbol{\xi}$, first, the pair $(\mathsf{a}_n, \mathsf{a}_{n+1})$ is added to the edge set $\widehat{E}$ if not already present; second, for each *unique* skill transition $(\mathsf{a}_n, \mathsf{a}_{n+1})$, a set of *augmented* states is collected, denoted by $\widehat{\mathbf{s}}_{\mathsf{a}_n\mathsf{a}_{n+1}} = \{\widehat{\mathbf{s}}\}$, where $\widehat{\mathbf{s}} = (\mathbf{s}_{n+1}, \mathbf{s}_G)$. Furthermore, for each augmented state $\widehat{\mathbf{s}}_\ell = (\mathbf{s}_t, \mathbf{s}_G) \in \widehat{\mathbf{s}}_{\mathsf{a}_n\mathsf{a}_{n+1}}$, we derive the following *feature* vector:

$$\mathbf{h}_\ell = (\mathbf{h}_{tG}, \mathbf{v}_G), \tag{8}$$

where $\mathbf{h}_{tG} = (\boldsymbol{H}_\mathsf{r}, \boldsymbol{H}_{\mathsf{o}_1}, \cdots, \boldsymbol{H}_{\mathsf{o}_H})$, where $\boldsymbol{H}_\mathsf{o} = (\mathbf{b}_\mathsf{o}, \boldsymbol{\alpha}_\mathsf{o}) \in \mathbb{R}^6$ is the relative translation and rotation of robot r and objects $\mathsf{o}_1, \mathsf{o}_2, \cdots, \mathsf{o}_H \in \mathsf{O}_{\mathsf{a}_n}$, from the current system state $\mathbf{s}_t$ to the goal state $\mathbf{s}_G$; $\mathbf{v}_G$ is the feature vector defined in (4) associated with the goal state $\mathbf{s}_G$. Note that $\mathbf{h}_\ell$ encapsulates features from both the relative transformation to the goal, and the goal state itself. Its dimension is linear to the number of objects relevant to skill a_n, as shown in Fig. 3.

Once all plans within $\boldsymbol{\Xi}$ are processed, the GTN $\mathcal{G}$ can be constructed as follows. First, its nodes and edges are directly derived from $\widehat{E}$. Then, for each node a, the set of its outgoing edges in $\widehat{E}$ is given by $\widehat{E}_\mathsf{a} = \{(\mathsf{a}, \mathsf{a}_\ell) \in \widehat{E}\}$. Thus the function $f(\mathsf{a})$ returns the edge selector $\mathcal{C}^{\mathrm{E}}_\mathsf{a}$ over $\widehat{E}_\mathsf{a}$. To compute this selector, we first construct the following training data:

$$\boldsymbol{\tau}^{\mathrm{E}}_\mathsf{a} = \left\{ (\mathbf{h}_\ell,\, e),\, \forall \widehat{s}_\ell \in \widehat{\mathbf{s}}_e,\, \forall e \in \widehat{E}_\mathsf{a} \right\}, \tag{9}$$

where e is the label for an edge $e = (\mathsf{a}, \mathsf{a}_\ell) \in \widehat{E}_\mathsf{a}$; $\mathbf{h}_\ell$ is the feature vector derived from (8). Then the edge selector $\mathcal{C}^{\mathrm{E}}_\mathsf{a}$ can be learned via the multi-nomial classification algorithms presented in Sec. III-A. Similar to (6), given a new scenario with state $\mathbf{s}_t$ and the specified goal state $\mathbf{s}_G$, the probability of choosing edge e is given by:

$$\rho_e = \mathcal{C}^{\mathrm{E}}_\mathsf{a}\left((\mathbf{s}_t,\, \mathbf{s}_G),\, e\right),\ \forall e \in \widehat{E}_\mathsf{a}, \tag{10}$$

where $\rho_e \in [0,\, 1]$. Note that ρ_e is trivial for skills with only one outgoing edge.

C. Human-in-the-loop Policy Learning and Execution

The previous sections present the methods to learn the extended skill model and the task network. The required training data are execution trajectories of the skill in (3) and complete plans of the task in (7). In this section, we show how human instructions during run time can be used to generate such data, and thus to learn or improve both the skill model and the task network *on-the-fly*.

1) Execute and Update GTN: The GTN $\mathcal{G}$ is initialized as empty. Consider a problem instance of the task, namely $(\mathbf{s}_0, \mathbf{s}_G)$. The system starts from state $\mathbf{s}_n$ whereas the GTN starts from the virtual start node $\mathsf{a}_n = \underline{\mathsf{a}}$ for $n = 0$. Then the associated edge selector $\mathcal{C}^{\mathrm{E}}_{\mathsf{a}_n}$ is used to compute the probability ρ_e of each outgoing edge $e \in \widehat{E}_{\mathsf{a}_n}$ based on (10). Then, the next skill to execute is chosen as:

$$\mathsf{a}^\star_{n+1} = \underset{e \in \widehat{E}_{\mathsf{a}_n}}{\operatorname{argmax}} \left\{ \rho_e(\mathbf{s}_n,\, \mathbf{s}_G),\, \text{where } \rho_e > \underline{\rho}^{\mathrm{E}} \right\}, \tag{11}$$

where $\underline{\rho}^{\mathrm{E}} > 0$ is a design parameter as the lower confidence bound. Note that if the current set of outgoing edges is empty, i.e., $\widehat{E}_{\mathsf{a}_n} = \varnothing$, or the maximal probability of all edges is less than $\underline{\rho}^{\mathrm{E}}$, the human operator is asked to *input* the preferred next skill $\mathsf{a}^\star_{n+1}$. Consequently, an *additional* data point is added to the training data $\boldsymbol{\tau}^{\mathrm{E}}_{\mathsf{a}_n}$, i.e.,

$$\boldsymbol{\tau}^{\mathrm{E}}_{\mathsf{a}_n} \leftarrow \left(\mathbf{h}(\mathbf{s}_n,\, \mathbf{s}_G),\, (\mathsf{a}_n,\, \mathsf{a}^\star_{n+1})\right), \tag{12}$$

where the feature vector $\mathbf{h}$ is computed based on (8). Thus, a new edge $(\mathsf{a}_n,\, \mathsf{a}^\star_{n+1})$ is added to the graph topology (V, E) if not present, and the embedded function $f(\cdot)$ is updated by re-learning the edge selectors $\mathcal{C}^{\mathrm{E}}_{\mathsf{a}_n}$ given this new data point.

2) Execute and Update Branch Selector: Now assume that a_{n+1} is chosen as the next skill. Then the branch selector $\mathcal{C}^{\mathrm{B}}_{\mathsf{a}_{n+1}}$ is used to predict the probability of each branch ρ_b, $\forall b \in B_{\mathsf{a}_{n+1}}$ Then, the most likely branch for a_{n+1} is chosen by:

$$b^\star_{n+1} = \underset{b \in B_{\mathsf{a}_{n+1}}}{\operatorname{argmax}} \left\{ \rho_b(\mathbf{s}_n),\, \text{where } \rho_b > \underline{\rho}^{\mathrm{B}} \right\}, \tag{13}$$

where $\underline{\rho}^{\mathrm{B}} > 0$ is another parameter as the lower confidence bound for the branch selection. Note that if no branch is found above, then the human operator is asked to input the preferred branch $b^\star_{n+1}$ for skill a_{n+1}. Consequently, an *additional* data point is added to the training data $\boldsymbol{\tau}^{\mathrm{B}}_{\mathsf{a}_{n+1}}$, i.e.,

$$\boldsymbol{\tau}^{\mathrm{B}}_{\mathsf{a}_{n+1}} \leftarrow \left(\mathbf{v}(\mathbf{s}_n),\, b^\star_{n+1}\right), \tag{14}$$

where the feature vector $\mathbf{v}$ is computed based on (4).

Algorithm 1: HIL Coordination of LfD Skills

Input: $\{\boldsymbol{D}_{\mathsf{a}}, \forall \mathsf{a} \in \mathsf{A}\}$, human inputs $\{\mathsf{a}_n^\star, b_n^\star\}$.
Output: $\mathcal{G}$, $\{\mathcal{C}_{\mathsf{a}}^{\mathrm{B}}\}$, $\boldsymbol{u}^\star$.
/* offline learning */
1 Learn $\boldsymbol{\theta}_{\mathsf{a}}$ and $\{\mathcal{C}_{\mathsf{a}}^{\mathrm{B}}\}$, $\forall \mathsf{a} \in \mathsf{A}$.
2 Initialize or load existing $\mathcal{G}$.
/* online execution and learning */
3 **while** *new task* $(\mathbf{s}_0, \mathbf{s}_G)$ *given* **do**
4 Set $\mathsf{a}_n \leftarrow \underline{\mathsf{a}}$ and $\mathbf{s}_n \leftarrow \mathbf{s}_0$.
5 **while** $\mathbf{s}_n \neq \mathbf{s}_G$ **do**
6 $\mathcal{G}$, $\mathsf{a}_{n+1} = \texttt{ExUpGtn}(\mathcal{G}, \mathsf{a}_n, (\mathbf{s}_n, \mathbf{s}_G), \mathsf{a}_{n+1}^\star)$.
7 $\mathcal{C}_{\mathsf{a}_{n+1}}^{\mathrm{B}}$, $b_{n+1} = \texttt{ExUpBrs}(\mathcal{C}_{\mathsf{a}_{n+1}}^{\mathrm{B}}, \mathbf{s}_n, b_{n+1}^\star)$.
8 Compute $\boldsymbol{u}^\star$ for branch b_{n+1} of skill a_{n+1}.
9 Obtain new state $\mathbf{s}_{n+1}$. Set $n \leftarrow n+1$.

3) Skill Execution: Once the optimal branch $b^\star$ is chosen for the desired next skill $\mathsf{a}_{n+1}^\star$, its trajectory can be retrieved given the skill model $\boldsymbol{\theta}_{\mathsf{a}_{n+1}}$. The retrieval process consists of two steps: First, the most-likely sequence of GMM components within the *desired branch* (denoted by $\boldsymbol{k}_T^\star$) can be computed via the modified Viterbi algorithm proposed in our earlier work [13]. Then, a reference trajectory is generated by an optimal controller (e.g., LQG from [28]) to track this sequence of Gaussians in the task space. Thus this reference trajectory is then sent to the low-level impedance controller to compute the control signal $\boldsymbol{u}^\star$. More algorithmic details and extension to Riemannian manifolds are given in [13].

Afterwards, the system state is changed to $\mathbf{s}_{n+2}$ with different poses of the robot and the objects, i.e., obtained from the state estimation and perception modules. Given this new state, the same process is repeated to choose the next skill and its optimal branch, until the goal state is reached.

D. Overall Algorithm

The overall procedure is summarized in Alg. 1. Namely, during the online execution for solving new tasks, Sec. V-C.1 is followed to execute and update the GTN, namely the function $\texttt{ExUpGtn}(\cdot)$ in Line 6, with possible human input $\mathsf{a}_n^\star$ if required. Once the next skill a_{n+1} is chosen, Sec. V-C.2 is followed to execute and update the branch selector, namely the function $\texttt{ExUpBrs}(\cdot)$ in Line 7, with possible human input $b_{n+1}^\star$ if required. Consequently the GTN and branch selectors are updated and improved via (12) and (14) on-the-fly. Compared with the manual specification of the transition and branching conditions, the required human inputs above are more intuitive and easier to specify.

Lastly, the computation complexity of the logistic regression is $\mathcal{O}(d^2)$, where d is the dimension of the feature vector. The skill model learning and the LQG control algorithm have polynomial complexity to the robot state dimension.

VI. Experiments

This section presents the experimental validation on a 7-DoF Franka Emika robot arm for two different industrial applications: first one as a part of an assembly task, and second as a bin-sorting task. The arm is extended by a Zivid RGBD sensor for perception and a parallel (or suction) gripper. Additionally, kinaesthetic teaching can be done directly by guiding the end effector manually. The proposed framework is implemented in Python3 under the Robot Operating System (ROS). All benchmarks are run on a desktop with an 8-core Intel Xeon CPU. Experiment videos can be found in the supplementary file.

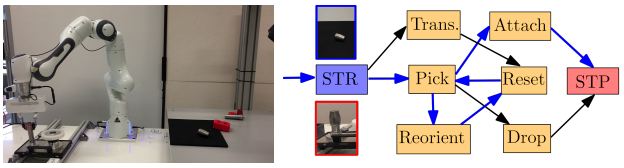

Fig. 5: **Left**: workspace for the assembly task; **Right**: the learned final GTN with one example plan (in blue).

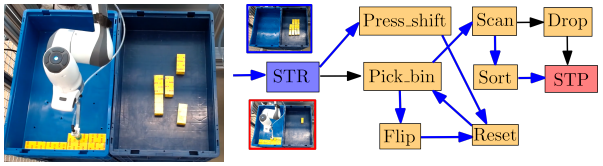

Fig. 6: **Left**: workspace for the bin-sorting task; **Right**: the learned final GTN with one example plan (in blue).

A. Workspace and Task Description

The assembly task was introduced in our previous work [13]. For brevity, we omit the detailed description here and refer the readers there. As shown in Fig. 5, a metallic cap is fed onto the inspection platform and depending on the result, the robot arm should either attach the cap to the top of a peg or drop it into a pallet. Given different initial states of the cap (lying-flat of standing), additional manipulation skills are needed to re-orient and translate the cap before the normal pick and drop skills. Note that in our earlier work [12], [13], these sequences are specified *manually* before every execution.

Another application is the well-known bin picking and sorting task. As shown in Fig. 6, the goal is to pick unknown objects out of the bin, scan them for product info, and then sort them accordingly. Instead of emphasizing performance such as "pick per hour", we are interested in addressing several corner cases: (i) the object should be picked with different orientations when close to the bin boundaries; (ii) the object should be cleared out of the corners before picking; (iii) the object should be flipped when the barcode overlaps with the suction area; and (iv) the objects are placed differently given the scanning results.

B. Results

1) Learned Skill Model: For the assembly application, in total 5 primitive skills are taught, i.e., `pick` to pick the cap from the platform with three branches; `re_orient` to re-orient the cap from lying flat to standing with two branches; `translate` to translate the cap to the platform boundary while standing; `attach` to attach the cap to the peg; `drop` to drop the cap with two branches. Similarly, for the bin

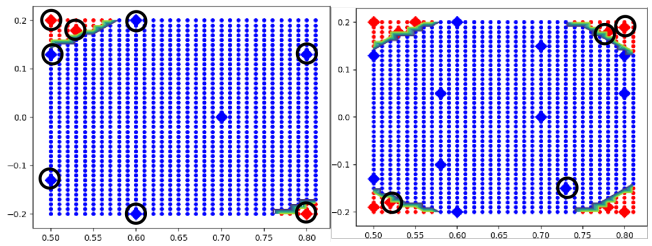

Fig. 7: Evolution of the edge selector for node `STR` in the GTN for bin-sorting in Fig. 6. There are two modes projected onto the $x-y$ plane: `pick_bin` skill (in red) and `press_shift` skill (in blue). Human instructions are requested at the samples marked in circles, while autonomous predictions are marked in diamonds.

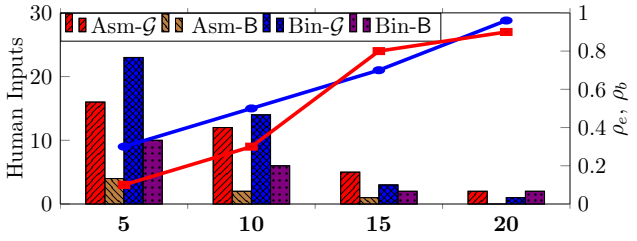

Fig. 8: x-axis: number of task executions. Left-y-axis: number of human instructions requested for GTN ($\mathcal{G}$) and branch selector (B), for assembly task `Asm` and bin-sorting task `Bin`. Right-y-axis: evolution of the lowest confidence in the GTN execution for both tasks.

sorting application, in total 6 primitive skills are taught, i.e., `pick_bin` to pick any object from bin with five branches. `scan` to scan the object; `press_shift` to press and shift any object out of the corners with four branches; `flip` to flip the object; `drop_bin` to drop the picked object into another bin; `sort` to arrange the picked object in rows.

Due to the new branch selector proposed in Sec. V-A, the number of demos needed is much reduced compared with our previous work [12], [13]. In average 6 demos are performed for each skill and the associated skill model with the branch selector is learned in $0.2s$. Afterwards, the model of each skill is verified independently under different scenes. For instance, the `pick` skill are demonstrated 2 times for each branch, and the feature vector $\mathbf{v}$ has dimension 7.

2) Learned Task Network: The proposed GTN is learned by following Alg. 1 for each application. Various problem instances of $(\boldsymbol{s}_0, \boldsymbol{s}_G)$ are defined in the learning process. Human instructions are requested for the next desired skill and branch. For instance, as shown in Fig. 5, if the cap is standing and the goal is to drop it into the pallet, the skill sequence is `pick`, `translate`, `pick` again, and finally `drop`. As shown in Fig. 6, if the object is close to one corner with the barcode on the top, the sequence is `press_shift`, `pick_bin`, `flip`, `scan`, and `drop` into another bin.

Each time a human instruction is given, either the GTN or the branch selector is updated via Line $6-7$ in Alg. 1. Fig. 7 shows an example how the edge selector of node GTN evolves within the execution of the first five executions. Moreover, Fig. 8 records how many human inputs are required for the GTN $\mathcal{G}$, and the branch selectors $\{\mathcal{C}^{\mathrm{B}}\}$ are updated during the whole run. Notice that the topology of $\mathcal{G}$ is quickly learned while the edge and branch selectors are improved whenever a new scene is experienced with a low confidence score. Fig. 8 also shows how the lowest confidences for selecting edges and branches increase with time as the GTN is improved, where both lower bounds $\underline{\rho}^{\mathrm{B}}$ and $\underline{\rho}^{\mathrm{E}}$ are set to 0.8. In the end, the success rate is close to 100% with full autonomy for both applications, where most failures are caused by execution and perception errors. The final learned GTNs for both are shown in Fig. 5 and 6.

Methods	L-time[s]	S-time[s]	R-skill/task	H-design[min]
GTN	1.5	0.2	0.95/0.9	2
TPH	0.1	0.8	0.8/0.6	N/A
FUL	10	0.5	0.3/0.2	10
TMP	N/A	>300	0.6/0.5	N/A
MAN	N/A	0.1	0.9/0.9	20

TABLE I: Comparison: learning time, solution time, success rate for each skill and the complete task, and the total time to design human inputs, of all baselines for the assembly task.

C. Comparison

The proposed scheme (**GTN** for short) is compared to the following baselines: (i) the vanilla TP-HSMM scheme as stated in [5], [13] (**TPH** for short), i.e., in combination with proposed GTN but without the proposed branch selector; (ii) the *full* system state (**FUL** for short) is used as the feature vector for the branch selector in (4) and the edge selector in (8), i.e., instead of the relative frames; (iii) a task and motion planner (**TMP** for short) that searches in simulation over the system state space for each new task $(\boldsymbol{s}_0, \boldsymbol{s}_G)$; (iv) a completely manual design of the branch and edge selection (**MAN** for short), i.e., by specifying the rules for each case.

As summarized in Table I, **TMP** does not learn from past solutions and requires the longest solution time. For certain problems where the sequence has more than 4 skills, it can not solve them in reasonable time (10 min). Moreover, **TPH** performs relatively well for predicting the skill sequence, however fails at executing the skill due to choosing the wrong branch, especially when the scenes are different from the demos. Notably, **FUL** learns not only slower but also performs worse in predicting both the edges and the branches, compared with **GTN**. The plausible explanation is that the *relative transformation* in (4) and (8) is difficult to capture with linear or even nonlinear kernels such as RBF [25]. Last but not least, the manual rules in **MAN** are much harder to design than the string inputs required by our scheme. Particularly, for both applications, boundaries on orientations of *different* objects are often transformed to Euler angles, which however are often ill-posed; In addition, such rules are hard to cover the complete state space and thus some corner cases are not defined.

VII. Conclusion

This work proposes a human-in-the-loop coordination framework for LfD skills that constructs and learns a geometric task network on-the-fly from human instructions. The resulting framework is data-efficient and intuitive even for non-technical operators. Future work involves the composition of various GTNs and interactive teaching.

REFERENCES

[1] S. M. LaValle, *Planning Algorithms*. Cambridge university press, 2006.

[2] M. Hermann, T. Pentek, and B. Otto, "Design principles for industrie 4.0 scenarios," in *Hawaii International Conference on System Sciences (HICSS)*, 2016, pp. 3928–3937.

[3] D. Xin, L. Ma, J. Liu, S. Macke, S. Song, and A. Parameswaran, "Accelerating human-in-the-loop machine learning: challenges and opportunities," in *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, 2018, pp. 1–4.

[4] A. Holzinger, "Interactive machine learning for health informatics: when do we need the human-in-the-loop?" *Brain Informatics*, vol. 3, no. 2, pp. 119–131, 2016.

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

[6] 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.

[7] D. Pathak, P. Mahmoudieh, G. Luo, P. Agrawal, D. Chen, Y. Shentu, E. Shelhamer, J. Malik, A. A. Efros, and T. Darrell, "Zero-shot visual imitation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 2050–2053.

[8] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2004, p. 1.

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

[10] L. Rozo, S. Calinon, D. G. Caldwell, P. Jimenez, and C. Torras, "Learning physical collaborative robot behaviors from human demonstrations," *IEEE Transactions on Robotics (TRO)*, vol. 32, no. 3, pp. 513–527, 2016.

[11] M. Zeestraten, "Programming by demonstration on Riemannian manifolds," 2017, phD thesis.

[12] L. Schwenkel, M. Guo, and M. Bürger, "Optimizing sequences of probabilistic manipulation skills learned from demonstration," in *Conference on Robot Learning (CoRL)*, 2019.

[13] L. Rozo, M. Guo, A. G. Kupcsik, M. Todescato, P. Schillinger, M. Giftthaler, M. Ochs, M. Spies, N. Waniek, P. Kesper, and M. Bürger, "Learning and sequencing of object-centric manipulation skills for industrial tasks," in *IEEE/RSJ International Conference on Intelligient Robots and Systems (IROS)*, 2020.

[14] A. T. Le, M. Guo, N. van Duijkeren, L. Rozo, R. Krug, A. G. Kupcsik, and M. Bürger, "Learning forceful manipulation skills from multimodal human demonstrations," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021.

[15] 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.

[16] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical planning in the now," in *AAAI Conference on Artificial Intelligence*, 2010.

[17] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, "From skills to symbols: Learning symbolic representations for abstract high-level planning," *Journal of Artificial Intelligence Research*, vol. 61, pp. 215–289, 2018.

[18] M. Toussaint, K. Allen, K. Smith, and J. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," in *Robotics: Science and Systems (R:SS)*, June 2018.

[19] D.-A. Huang, S. Nair, D. Xu, Y. Zhu, A. Garg, L. Fei-Fei, S. Savarese, and J. C. Niebles, "Neural task graphs: Generalizing to unseen tasks from a single video demonstration," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 8565–8574.

[20] B. Hayes and B. Scassellati, "Autonomously constructing hierarchical task networks for planning and human-robot collaboration," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 5469–5476.

[21] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto, "Robot learning from demonstration by constructing skill trees," *The International Journal of Robotics Research (IJRR)*, vol. 31, no. 3, pp. 360–375, 2012.

[22] E. Frazzoli, M. A. Dahleh, and E. Feron, "Maneuver-based motion planning for nonlinear systems with symmetries," *IEEE Transactions on Robotics (TRO)*, vol. 21, no. 6, pp. 1077–1091, 2005.

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

[24] T. Mandel, Y.-E. Liu, E. Brunskill, and Z. Popović, "Where to add actions in human-in-the-loop reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

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

[26] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society*, vol. 39, no. 1, pp. 1–38, 1977.

[27] M. Guo and M. Bürger, "Geometric task networks: Learning efficient and explainable skill coordination for object manipulation," *IEEE Transactions on Robotics*, 2021.

[28] L. Sciavicco and B. Siciliano, *Modelling and Control of Robot Manipulators*. Springer Science & Business Media, 2012.