Supplementary File for “Geometric Task Networks: Learning Efficient and Explainable Skill Coordination for Object Manipulation”

Meng Guo and Mathias Bürger.

I. Learning of Skill Model

A. Demonstrations

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 $\mathsf{O_a} \subseteq \mathsf{O}$ and the set of demonstrations is given by $\mathsf{D_a} = \{\mathsf{D}_1, \cdots, \mathsf{D}_{M_\mathsf{a}}\}$, where each demonstration D_m is a sequence of states s that consists of the robot end-effector state $\boldsymbol{r}$ within the manifold $\mathcal{M}_r$, and object states $\{\boldsymbol{p}_\mathsf{o}, \mathsf{o} \in \mathsf{O_a}\}$ each within the manifold $\mathcal{M}_p$, i.e.,

$$\mathsf{D}_m = \big[\mathsf{s}_t\big]_{t=1}^{T_m} = \big[\big(\boldsymbol{r}_t,\, \{\boldsymbol{p}_{t,\mathsf{o}},\, \mathsf{o} \in \mathsf{O_a}\}\big)\big]_{t=1}^{T_m}. \tag{1}$$

Via a combination of these skills, the objects can be manipulated to reach a desired state.

B. Trajectory Model

1) TP-GMM: 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 $\boldsymbol{r} = \{\boldsymbol{r}_t\}_{t=1}^N$, where $\boldsymbol{r}_t \in \mathbb{R}^d$. 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 $\boldsymbol{r}_t^{(p)} = \boldsymbol{A}^{(p)^{-1}}(\boldsymbol{r}_t - \boldsymbol{b}^{(p)})$. 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 in [1], 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. The Expectation-Maximization (EM) algorithm from [2] is a well-established method to learn such models.

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

$$\begin{aligned}\hat{\boldsymbol{\Sigma}}_k &= \left[\sum_{p=1}^P \left(\hat{\boldsymbol{\Sigma}}_k^{(p)}\right)^{-1}\right]^{-1}, \\ \hat{\boldsymbol{\mu}}_k &= \hat{\boldsymbol{\Sigma}}_k \left[\sum_{p=1}^P \left(\hat{\boldsymbol{\Sigma}}_k^{(p)}\right)^{-1} \hat{\boldsymbol{\mu}}_k^{(p)}\right],\end{aligned} \tag{2}$$

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 $\hat{\boldsymbol{\Sigma}}_k^{(p)} = \boldsymbol{A}^{(p)}\boldsymbol{\Sigma}_k^{(p)}\boldsymbol{A}^{(p)^\intercal}$. While the task parameters may vary over time, we dropped the time index for the sake of notation. More mathematical derivations can be found in [3].

2) TP-HSMM: 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 underlying hidden process is assumed to be Markov, i.e., the probability of transitioning to the next state depends only on the current state, in HSMM the state process is assumed semi-Markov. This means that a transition to the next state depends on the current state as well as on the elapsed time since the state was entered. They have been successfully applied, in combination with TP-GMMs, for robot skill encoding to learn spatio-temporal features of the demonstrations from [4]. 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 $\{\boldsymbol{r}_\ell\}_{\ell=1}^t$, assume that the associated sequence of states in $\boldsymbol{\Theta}$ is given by $\mathsf{s}_t = s_1 s_2 \cdots s_t$, where t is the total length. As shown in [4], the probability of data point $\boldsymbol{r}_t$ belonging

Bosch Center for Artificial Intelligence (BCAI), Germany. Corresponding author: `Meng.Guo2@de.bosch.com`

translate $\mathsf{a}_{\mathtt{tl}}$		insert $\mathsf{a}_{\mathtt{is}}$	
$\mathsf{TP}_{\mathtt{t}}$	Success Rate	$\mathsf{TP}_{\mathtt{i}}$	Success Rate
$\{\mathsf{d},\mathsf{r}\}$	0.08	$\{\mathsf{b}_1,\mathsf{r}\}$	0.05
$\{\mathsf{d},\mathsf{r},\mathsf{o}\}$	0.26	$\{\mathsf{b}_1,\mathsf{r},\mathsf{o}\}$	0.44
$\{\mathsf{d},\mathsf{o}\}$	0.37	$\{\mathsf{b}_1,\mathsf{o}\}$	0.71
$\{\mathsf{r},\mathsf{o}\}$	0.99	$\{\mathsf{r},\mathsf{o}\}$	0.99

TABLE I
THE CHOICE OF TASK PARAMETERS AND ASSOCIATED SUCCESS RATE, FOR SKILLS TRANSLATE AND INSERT USED IN THE EXPERIMENT. NOTE THAT "r" STANDS FOR THE INITIAL POSE OF THE ROBOT ARM, "o" THE INITIAL OBJECT POSE, "b$_1$" THE POSE OF THE CONTAINER TO INSERT THE OBJECT, AND "d" THE DESTINATION POSE OF THE OBJECT OF THE TRANSITION SKILL TRANSLATE.

to state k (i.e., $s_t = k$) is given by the *forward* variable $\alpha_t(k) = p(s_t = k, \{\boldsymbol{r}_\ell\}_{\ell=1}^t)$:

$$\alpha_t(k) = \sum_{\tau=1}^{t-1}\sum_{h=1}^{K} \alpha_{t-\tau}(h) a_{hk}\, \mathcal{N}(\tau|\mu_k^D, \sigma_k^D)\, o_\tau^t\,, \tag{3}$$

where $o_\tau^t = \prod_{\ell=t-\tau+1}^{t} \mathcal{N}(\boldsymbol{r}_\ell|\hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k)$ is the emission probability and $(\hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k)$ are derived from (2) 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 [5], i.e., by setting $\mathcal{N}(\boldsymbol{r}_\ell|\hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k) = 1$ for all k and $\ell > t$ in (3).

$$\boldsymbol{\theta}_{\mathsf{a}} = \Big\{\{a_{kh}\}_{h=1}^K, (\mu_k^D, \sigma_k^D), \{\pi_k, \{(\boldsymbol{\mu}_k^{(\mathsf{p})}, \boldsymbol{\Sigma}_k^{(\mathsf{p})})\}_{\mathsf{p}\in\mathsf{TP}_{\mathsf{a}}}\}\Big\}_{k=1}^K .$$

Details about TP-HSMMs and its usage as skill representation can be found in [6]. At last, the sequence of the most-likely states $\mathbf{s}_{T_m}^\star = s_1^\star s_2^\star \cdots s_{T_m}^\star$ is determined by choosing $s_t^\star = \arg\max_k \alpha_t(k)$, $\forall 1 \leqslant t \leqslant T_m$.

3) Choice of Task Parameters: The success of both learning and reproducing skills with TP-GMMs heavily depends on a *good* choice of task parameters, which the user has to provide. As shown in Table I, different choices of task parameters can result in significant changes in the performance. For example, a proper choice of task parameters for the skill `grasp_peg` is a coordinate frame attached to the peg and one at the initial pose of the robot arm. This allows the learned trajectory to have a smooth start and more importantly adapt to new poses of the peg for successful grasping. As rule of thumb, attaching frames to all involved objects $\mathsf{O_a}$ and to the robot arm initial pose indexed by r as well as using the free task parameters $\mathsf{F_a}$ for transition skills, i.e. $\mathsf{TP_a} = \mathsf{O_a} \cup \mathsf{F_a} \cup \{\mathsf{r}\}$, covers many cases. However, this is not always the best choice, since some objects might produce irrelevant task parameters, which not only increases the computation cost but can also decrease the performance of reproduction. A problem that arises with time-varying task parameters like an object pose is that the TP-HSMM only encodes how the task parameter influences the robot arm motion, but not how the robot arm motion affects the objects pose. For example, while executing the skill `translate_object` the trajectory of the robot arm in the frame attached to the object is only a single constant point, because the object follows every motion of the robot arm while it is grasped. Thus, the robot arm will follow the object during reproduction, i.e. stay where the object is, since the trajectory generation does not know that the robot arm can be moved freely without leaving the single point component in the object frame. In this case, it is better to not to use the object frame as task parameter.

In order to automate the choice of a proper set of task parameters $\mathsf{TP_a} \subseteq \mathsf{O_a} \cup \mathsf{F_a} \cup \{\mathsf{r}\}$, we can validate a choice by computing its reproduction error. For this we need a ground truth, which is already available as human demonstrations. Usually, the set of demonstrations $\mathsf{D_a}$ is rather small, such that we have to use the same set of demonstrations $\mathsf{D_a}$ for training and validation. This yields to the validation error:

$$V(\mathsf{TP_a}) = \sum_{m=1}^{M_{\mathsf{a}}}\sum_{t=1}^{T_m} \big\|\mathrm{Log}_{\boldsymbol{r}_t}(\hat{\boldsymbol{r}}_t)\big\|\,, \tag{4}$$

where $\hat{\boldsymbol{r}}_t$ is the trajectory retrieved from $\boldsymbol{\theta}_{\mathsf{a}}(\mathsf{TP_a})$ for the task parameters from demonstration D_m.

The number of involved objects for a skill is usually small, then we can train the model for all combinations of task parameters and validate each choice. If the number of objects is higher, the user has to preselect some promising choices of task parameters to reduce the computation time. The examples in Table I for the skills `translate` (object "o" to destination "d") and `insert` (object "o" into box "b") show that in both cases it is better to not use the object frame as task parameter due to the reasons described above. The choices of not using the robot initial arm position "r" or the destination "d" or box "b" are as expected to be even worse.

C. Precondition and Effect Model

The precondition of a skill refers to the relative relations between the robot arm and the relevant objects, which should be satisfied initially for the skill execution to be successful. The effect of a skill refers to how the skill execution would change the state. In this part, we describe how to learn these models purely from the available demonstrations.

1) Task Parameterized Model: The trajectory model $\boldsymbol{\theta}_{\mathsf{a}}$ does not incorporate how the objects or robot arm are located w.r.t. each other when the skill execution starts and finishes. The proposed idea is to learn task-parameterized Gaussians (TP-Gs) for each object to fit its pose from demonstrations. The *precondition* model is:

$$\gamma_{1,\mathsf{a}}(\mathbf{s}, \boldsymbol{p}_{\mathsf{F}}) \triangleq \Big\{(\boldsymbol{\mu}_{1,\mathsf{o}}^{(\mathsf{p})}, \boldsymbol{\Sigma}_{1,\mathsf{o}}^{(\mathsf{p})}),\ \forall \mathsf{p} \in \mathsf{TP_a}\backslash\{\mathsf{o}\}\Big\}_{\mathsf{o}\in\mathsf{O_a}\cup\mathsf{F_a}}\,, \tag{5}$$

where $\boldsymbol{p}_{\mathsf{F}}$ is the choice of free task parameters; $\mathsf{F_a}$ is the set of free task parameters; $(\boldsymbol{\mu}_{1,\mathsf{o}}^{(\mathsf{p})}, \boldsymbol{\Sigma}_{1,\mathsf{o}}^{(\mathsf{p})})$ is the Gaussian distribution of object o's initial pose at time 1 from the perspective of object p's initial pose at initial time 1. Thus it is also called the "initial-to-initial" precondition model. Similarly, the *effect* model of a skill is defined by:

$$\gamma_{T,\mathsf{a}}(\mathbf{s}, \boldsymbol{p}_{\mathsf{F}}) \triangleq \Big\{(\boldsymbol{\mu}_{T,\mathsf{o}}^{(\mathsf{p})}, \boldsymbol{\Sigma}_{T,\mathsf{o}}^{(\mathsf{p})}),\ \forall \mathsf{p} \in \mathsf{TP_a}\Big\}_{\mathsf{o}\in\mathsf{O_a}}\,, \tag{6}$$

where $\boldsymbol{p}_{\mathsf{F}}$ is the choice of free task parameters; $(\boldsymbol{\mu}_{T,\mathsf{o}}^{(\mathsf{p})},\, \boldsymbol{\Sigma}_{T,\mathsf{o}}^{(\mathsf{p})})$ is the Gaussian distribution of object o's final pose at time T from the perspective of object p's initial pose. Thus it is also called the "initial-to-final" effect model. Note that both models are computed within the object pose manifold $\mathcal{M}_p$. Low variance in the learned models indicate a consistent geometric relation in the demonstrations, e.g., the precondition of skill `insert` is that the object is grasped by arm initially, while the effect of skill `translate` is that the object is put at the destination. For example, the precondition model of skill `insert` requires very low variance between the initial poses of robot arm "`r`" and object "`o`" as they always satisfy the grasping relation. The effect model of skill `translate` requires very low variance between the final pose of object `o` and the initial pose of destination `d` as the object is always put on top of it.

2) Evaluation of Precondition: Given a new system state s and a choice of free task parameters $\boldsymbol{p}_{\mathsf{F}}$, we can evaluate how much the precondition of a skill is satisfied using the learned models. In particular, we can compute the product of the observation probability for the robot arm and each object, or equivalently the logsum:

$$c_{\mathsf{a}}(\mathrm{s},\, \boldsymbol{p}_{\mathsf{F}}) \triangleq \log\left(\sum_{k=1}^{K} \pi_k\, \mathcal{N}(\boldsymbol{r}\,|\,\hat{\boldsymbol{\mu}}_k,\, \hat{\boldsymbol{\Sigma}}_k)\right) + \sum_{\mathsf{o}\in \mathsf{O}_{\mathsf{a}}\cup \mathsf{F}_{\mathsf{a}}} \log\left(\mathcal{N}(\boldsymbol{p}_{\mathsf{o}}\,|\,\hat{\boldsymbol{\mu}}_{1,\mathsf{o}},\, \hat{\boldsymbol{\Sigma}}_{1,\mathsf{o}})\right), \tag{7}$$

where $\{(\hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k)\}$ are the combined Gaussians of initial robot arm pose in the global frame from the learned trajectory model $\boldsymbol{\theta}_{\mathsf{a}}$; $\{(\hat{\boldsymbol{\mu}}_{1,\mathsf{o}}, \hat{\boldsymbol{\Sigma}}_{1,\mathsf{o}})\}$ are the combined Gaussians of object o's initial pose in the global frame from the learned precondition model $\gamma_{1,\mathsf{a}}$. The computation of these components involves transforming Gaussians from local frames to the global frame and then computing their product. No closed analytical forms exist for general Riemannian manifolds, see [7]. Note that the measure above is not a probability, but computation over probability densities. It provides a continuous value that evaluates how similar the current situation is to the demonstrations. This measure can be already useful to: (a) to decided whether the precondition of skill a is satisfied by comparing with a given threshold $\underline{c}$, i.e., $c_{\mathsf{a}}(\cdot) > \underline{c}$; (b) to compare different scenarios and different free task parameters.

3) Prediction of Effect: The effect includes the poses of both robot arm and objects after executing the skill. First, the final pose of the robot arm follows the learned trajectory model $\boldsymbol{\theta}_{\mathsf{a}}$, i.e., $\boldsymbol{r}_T\,|\,(\mathrm{s}_0, \boldsymbol{p}_{\mathsf{F}}) \sim \mathcal{N}(\hat{\boldsymbol{\mu}}_K,\, \hat{\boldsymbol{\Sigma}}_K)$, where $(\hat{\boldsymbol{\mu}}_K,\, \hat{\boldsymbol{\Sigma}}_K)$ is directly the K-th combined Gaussian the global frame. Second, the final poses of all objects follow the learned effect model $\gamma_{T,\mathsf{a}}$, i.e., $\boldsymbol{p}_{T,\mathsf{o}}\,|\,(\mathrm{s}_0, \boldsymbol{p}_{\mathsf{F}}) \sim \mathcal{N}(\hat{\boldsymbol{\mu}}_{T,\mathsf{o}},\, \hat{\boldsymbol{\Sigma}}_{T,\mathsf{o}})$, where $(\hat{\boldsymbol{\mu}}_{T,\mathsf{o}},\, \hat{\boldsymbol{\Sigma}}_{T,\mathsf{o}})$ is the combined Gaussian of object o in the global frame. Thus, the estimated final state $\hat{\boldsymbol{s}}_T$ after executing skill a is given by:

$$\begin{aligned}\hat{\boldsymbol{s}}_T &\triangleq \Omega_{\mathsf{a}}(\mathrm{s}_0, \boldsymbol{p}_{\mathsf{F}})\\ &\triangleq \Big(\boldsymbol{r}_T\,|\,(\mathrm{s}_0,\, \boldsymbol{p}_{\mathsf{F}}),\; \{\boldsymbol{p}_{T,\mathsf{o}}\,|\,(\mathrm{s}_0,\, \boldsymbol{p}_{\mathsf{F}}), \mathsf{o}\in \mathsf{O}_{\mathsf{a}}\}\Big),\end{aligned} \tag{8}$$

where the mean of the corresponding Gaussians can be directly used as the most likely prediction. Different from the symbolic representations for planning from [8], the precondition and effect models learned in (7) and (8) are continuous and adaptive to the actual scenario during execution.

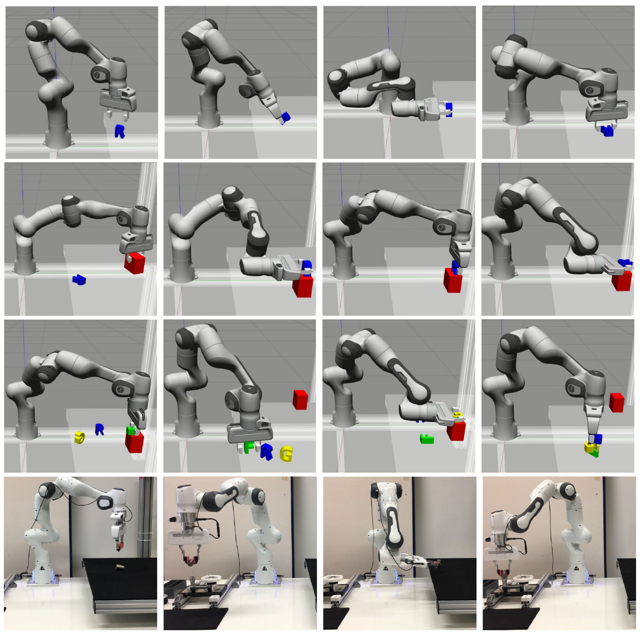
Fig. 1. Snapshots of executions for **Task-1,2,3** in Sec. II-A and **Assembly** task in Sec. II-B.

II. Details of Numerical Evaluation

This section contains the numerical validation for the proposed approach over a 7-DoF robotic manipulator, both in simulation and on actual hardware. Various tasks are addressed that require scene re-construction and tool-usage. The framework is implemented on top of Robot Operating System (ROS) to enable communication across planning, motion control and perception modules.

A. 6D Scrabble in Simulation

To avoid limitations from perception, we evaluate first various tasks in simulation where many objects are added freely into the planning scene. Particularly, several 3D letters are scattered on a platform along with a 7 DoF Franka Emika Panda robot from [9] with a two-finger gripper, as shown in Fig. 1. The alphabets are in average of size $6\,\mathrm{cm} \times 3\,\mathrm{cm} \times 6\,\mathrm{cm}$. The platform has a height $15\,\mathrm{cm}$ w.r.t. the table surface. We consider the following three progressive manipulation tasks with increasing complexity:

Task-1 Manipulate one alphabet to reach the desired position and orientation (i.e., pose in 6D) on the platform surface. Note that when the initial and the goal states of the alphabet are consistent (i.e., both standing or lying flat), it is enough to simply rotate and translate the alphabet according to the desired pose. However, if they are not consistent, then a re-orientation of the alphabet from standing to lying flat or vice versa is needed. This task is designed to show the computational complexity of a TAMP problem even for a simple task.

Skill Name	M_{a}	K_{a}	TP_{a}	$t\,(\theta_{\mathsf{a}}\,\vert\,\gamma_{\mathsf{a}})$ [s]
`Pi_St`	12	7	$\{A, R\}$	3.6 | 2.7
`Pi_Si`	10	8	$\{A, R\}$	4.2 | 3.1
`Pi_Fl`	8	7	$\{A, R\}$	3.8 | 3.2
`Re_St2Fl`	9	8	$\{A, R, A_G\}$	4.3 | 2.1
`Re_Fl2St`	10	8	$\{A, R, A_G\}$	3.8 | 2.2
`Ro_St`	15	6	$\{A, R, A_G\}$	4.2 | 2.7
`Ro_Fl`	10	6	$\{A, R, A_G\}$	3.3 | 2.2
`Tr_St`	10	7	$\{A, R, A_G\}$	4.1 | 1.9
`Tr_Fl`	10	7	$\{A, R, A_G\}$	4.3 | 2.1
`Reset`	7	6	$\{R\}$	2.8 | 1.5

TABLE II
DEMONSTRATED SKILLS A, NUMBER OF DEMONSTRATIONS M_{a}, NUMBER OF COMPONENTS K_{a}, CHOICE OF TASK PARAMETERS TP_{a}, AND TRAINING TIME FOR θ_{a} AND γ_{a}. THE FRAMES A, R, A_G STAND FOR ALPHABET, ROBOT AND THE ALPHABET GOAL, RESPECTIVELY.

Task-2 Repeat **Task-1** but with a levitation box. In this case, the box is used as a tool for the re-orientation, i.e., the box is used to levitate the alphabet such that re-orientation is possible at the platform center. This change seems subtle but significantly increases the difficulty of the task, as the box is not directly related to or reflected in the goal but used as a tool to accomplish the goal.

Task-3 Repeat **Task-2** but with *three* alphabets. In this case, the box is used as a shared tool among all the alphabets. As the planning goal, the alphabets can be directly lying or standing on the platform surface or stacked on top of each other. In the latter case, the order of manipulating the alphabets plays an important role, e.g., the alphabet at the bottom has to be manipulated first. Besides, how and when the box should be used in between depends on how many alphabets need re-orientation.

1) Primitive Skills: In this section, we give a detailed description of the set of primitive skills used in this simulation study that relevant to the tasks described above. As summarized in Table II, there are 10 primitive skills for each alphabet relevant to the above tasks:

- `Pi_St` and `Pi_Si` to pick the standing alphabet from the top or the side while `Pi_Fl` to pick a flat-lying alphabet;
- `Re_St2Fl` re-orients the alphabet from standing to flat while `Re_Fl2St` does the opposite;
- `Ro_St` and `Ro_Fl` rotate the alphabet by arbitrary *yaw* angle while standing and lying flat (without changing positions), respectively;
- `Tr_St` and `Tr_Fl` translate the alphabet to arbitrary position while standing and lying flat (without changing orientations), respectively;
- `Reset` resets the robot to default position.

Instead of kinesthetic teaching, we use existing motion planners such as RRTC [10] and manually-chosen waypoints to generate demonstrations under various poses of the robot and the alphabets. Demonstrations are recorded at $50\,\mathrm{Hz}$, where the state of the robot and the alphabets are fetched directly from the simulator. The number of demonstrations for each skill is shown in Table II.

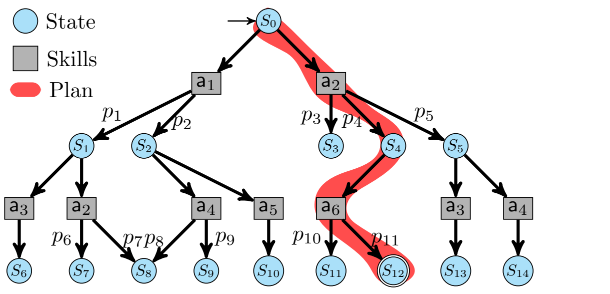

Fig. 2. Illustration of the TAMP setup for the hybrid search over the state graph.

It is worth mentioning that to avoid collision between the robot arm and the platform surface, the robot can *not* pick the alphabet from the side via the skill `Pi_Si`, while it is standing in the middle of the platform surface. Rather, the skill `Pi_Si` can only be executed when the alphabet is on the edge of the platform *or* on top of the box. For the same reason, the robot can not pick the alphabet from the top via the skill `Pi_St`, while it is lying flat, but only by the skill `Pi_Fl`. Some of the above skills can be shared across different alphabets and even the box with minor changes such as gripper closing-width.

2) Hybrid TAMP: The TAMP as a hybrid search planner over the state graph is illustrated in Fig. 2 and setup as follows. Starting from the initial state s_0, a graph search algorithm is used to traverse the state space until the goal state is reached, e.g., breadth-first search or Dijkstra or $A^\star$ algorithm [10]. Each skill is used to drive the system to a new state, which can be either computed by the effect model $\gamma_{T,\mathsf{a}}$ for skill representations learned in LfD, or via simulating its execution in the simulator for other general representations. The cost of the associated edge can be either set uniformly, or computed based the confidence measure proposed in [6], or even learned in a supervised manner as from [11]. If this new state is *close* enough to the goal state, it is marked as "goal". However the search continues as there could be multiple states close to the goal state with different costs. After a upper bound on the search time is reached, the *shortest* path from the initial state to the set of reached goal states is returned as the desired plan:

$$\boldsymbol{\xi} = \mathsf{s}_0\,(\mathsf{a}_0, \mathsf{TP}_{\mathsf{a}_0})\,\mathsf{s}_1\,(\mathsf{a}_1, \mathsf{TP}_{\mathsf{a}_1})\mathsf{s}_2 \cdots \mathsf{s}_N, \tag{9}$$

which includes not only the visited states but also the skill and the associated skill parameters.

More importantly, the task parameters of some skills can *not* be computed directly from the current state but need to be specified. The same skill with different parameters drives the system to different states. Thus, we uniformly sample from the Gaussian distribution given the precondition model $\gamma_{1,\mathsf{a}}$ of the skill representations learned from LfD. For instances, the task parameter associated with frame A_G for skill `Tr_St` has a distribution over $x \in [0.23, 0.39]$ and $y \in [0.33, 0.55]$, for skill `Ro_St` has a distribution over $yaw \in [-\pi, \pi]$. Similar statements hold for skills `Tr_Fl` and `Ro_Fl`. Thus, each time these skills are used to expand the search, the associated parameters are sampled accordingly.

Different choices of the cost function and the sampling strategy have a great impact on the time complexity and performance of the search method. Specifically, the uniform

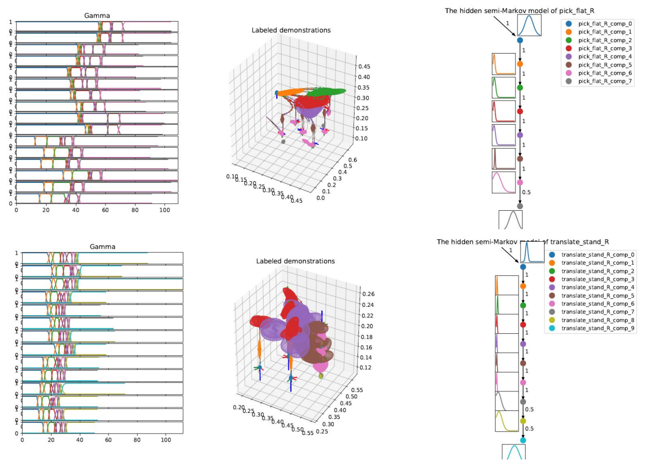

Fig. 3. Examples of skills for **Task-1** described in Sec. II-A3. Top: the learned model for skill `Pi_Fl`; Bottom: the learned model for skill `Tr_St`.

cost function allows faster exploration but for a larger part of the state space, while the confidence-guided search is slight slower to compute but for a much smaller part of the state space. Good heuristic for these aspects can boost the search efficiency significantly. Even manual bias in the sampling strategy can significantly reduce the number of samples and thus reduce the search space.

3) Task-1: Learning and Execution Results: First of all, the skill model $\mathcal{M}_{\mathsf{a}}$ is learned for each skill, of which the details such as the training time and the choice of frames are given in Table II. On average, 9 demonstrations are needed for each skill and the model is learned within 5 s. To give some examples, the learned skill models of `Pi_Fl` and `Tr_St` are shown in Fig. 3.

For a simple task such as **Task-1**, the desired behavior is already quite complex:

- If the initial and goal orientations of the alphabet are the same (i.e., both standing or lying flat), then the robot needs to pick the alphabet with the correct pick skill, rotate it to the desired orientation and translate it to the desired position. More specifically,
  - when the alphabet is initially standing, the desired skill sequence is `Pi_St` → `Ro_St` → `Tr_St` → `Reset`
    or `Pi_St` → `Tr_St` → `Ro_St` → `Reset`.
  - when the alphabet is initially lying flat, the desired skill sequence is `Pi_Fl` → `Ro_Fl` → `Tr_Fl` → `Reset`
    or `Pi_Fl` → `Tr_Fl` → `Ro_Fl` → `Reset`.
- However, if the initial and goal orientations of the alphabet are different (i.e., from standing to lying or vice versa), then the robot should re-orient the alphabet from standing to lying or vice versa. Note that since skill `Re_St2Fl` requires the alphabet to be grasped from the side and the skill `Pi_Si` can *only* be executed when the alphabet is standing on the platform edge, the alphabet should be translated to the edge by skill `Tr_St` before the re-orientation.
  - when the alphabet is initially standing and lying flat as goal, the desired skill sequence is `Pi_St` → `Ro_St` → `Tr_St` → `Reset` → `Pi_Si` → `Re_St2Fl` → `Ro_Fl` → `Tr_Fl` → `Reset`
    or `Pi_St` → `Tr_St` → `Ro_St` → `Reset` → `Pi_Si` → `Re_St2Fl` → `Ro_Fl` → `Tr_Fl` → `Reset`
    or `Pi_St` → `Tr_St` → `Ro_St` → `Reset` → `Pi_Si` → `Re_St2Fl` → `Tr_Fl` → `Ro_Fl` → `Reset`
    or `Pi_St` → `Ro_St` → `Tr_St` → `Reset` → `Pi_Si` → `Re_St2Fl` → `Tr_Fl` → `Ro_Fl` → `Reset`.
  - when the alphabet is initially lying flat and standing as goal, the desired skill sequence is `Pi_Fl` → `Re_Fl2St` → `Reset` → `Pi_St` → `Tr_St` → `Ro_St` → `Reset`
    or `Pi_Fl` → `Re_Fl2St` → `Reset` → `Pi_St` → `Ro_St` → `Tr_St` → `Reset`.

As a result, despite of its simplicity, **Task-1** requires 4 possible discrete sequences with length 4, 4, 9 or 7, of which contains 2, 2, 4 and 2 skill parameters (*each* as 6D poses).

We generated 200 problems of **Task-1**, half of which for training and another half for validation. The problems are generated by randomly sampling from possible poses of the alphabets (for initial and goal poses), with a equal distribution over the above four different cases. Then, these problems are solved by the hybrid TAMP described in Sec. II-A2. It took in total 7.3 h (in average 4 minutes per problem) to generate 100 successful plans as in (9), using the exhaustive hybrid TAMP mentioned above. As described in the learning algorithm of GTN, the associated GTN is learned in 2.8 s with 12 nodes and 20 edges (with 28 embedded TPGMMs), as shown in Fig. 4. Recall that the number of GMM components on each edge depends on the number of times this edge appears on the unique successful plans. For instance, the edge $(\texttt{Start}, \texttt{Pi\_St})$ has six components as it belongs six unique plans listed earlier, while the edge $(\texttt{Ro\_St}, \texttt{Tr\_St})$ has four components as it belongs to four unique plans listed earlier. The learned GTN topology shown in Fig. 4 matches the anticipated plans well. In particular, all anticipated cases are encapsulated in the learned GTN, e.g., all pick skills are allowed after reset; and the combination of rotation and translation skills to change the alphabet towards the goal state. translation or rotation skills are used to change the alphabet state before the reset. In addition, the embedding function $f(\cdot)$ for each edge takes in average 0.1 s to compute, which can also be intuitively explained, e.g., the TP-GMMs associated with the transition from `Pi_St` to `Ro_St` indicate that the alphabet should be rotated in yaw angle to align with the goal state (with small covariances) and the alphabet does not need to re-oriented. It is worth mentioning that such topology and geometric embeddings are learned *automatically* without any manual tunning. Afterwards, the learned GTN is used to solve the 100 problems in the validation set by following the execution procedure. Given the problem definition including the initial and goal states, the learned GTN can directly output the best next skill and the associated skill parameter. As a result, only the states and skills on the optimal path are

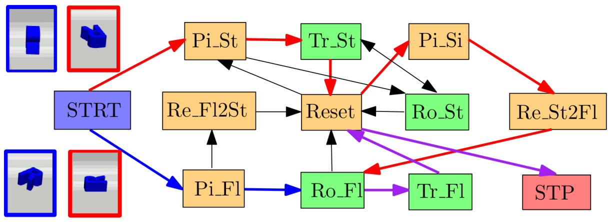

Fig. 4. The GTN associated with **Task-1**, which has 12 nodes and 20 edges (with 28 embedded TPGMMs).

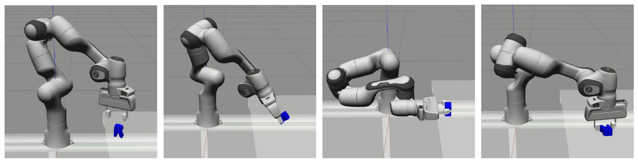

Fig. 5. Snapshots of execution of one case for **Task-1** in Sec. II-A3.

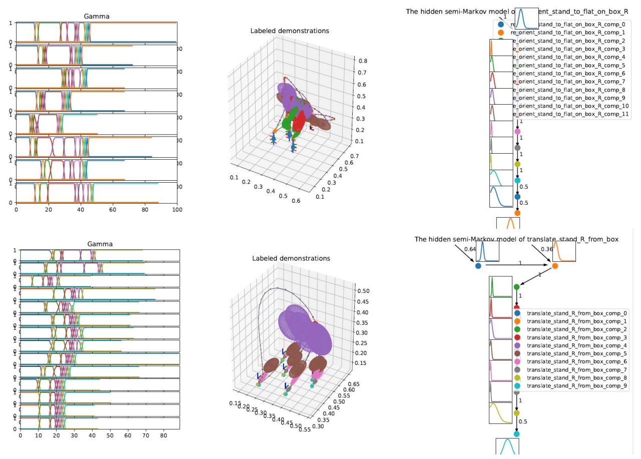

Fig. 6. Examples of additional skills for **Task-2** described in Sec. II-A4. Top: learned model for skill `Re_St2Fl_Bx`; Bottom: learned model for skill `Tr_St_Bx`.

explored and thus significantly reduce the planning time. One example of the execution under the learned GTN is shown in Fig. 5. More specifically, it took in total $10\,\text{min}$ (in average $6\,\text{s}$ per problem) to solve all problems in the validation set with a success rate of 100%, resulting in a 100-fold decrease in planning time while maintaining the same success rate as in training data. It can be seen that not only the correct edge is chosen but also the correct component on the embeddings is activated along the generated plan.

4) Task-2: Learning and Execution Results: In addition to the skills mentioned in **Task-1**, four new skills are demonstrated and added to the skill set to facilitate the re-orientation and translation of the alphabet between the platform surface and the box top:

- `Re_Fl2St_Bx`: to re-orient the lying-flat alphabet from the platform surface onto the top of the box and becoming standing. This skill is different from the existing skill `Re_Fl2St` as the box requires the motion to be higher in the z-axis.
- `Re_St2Fl_Bx`: to re-orient the standing alphabet from the platform surface onto the top of the box and becoming lying-flat. Note this skill is different from the existing skill `Re_St2Fl`.
- `Tr_St_Bx`: to translate the alphabet from the top of the box onto the surface of the platform while being standing. Note this skill is different from the existing skill `Tr_St`.
- `Tr_Fl_Bx`: to translate the alphabet from the top of the box onto the surface of the platform while being lying-flat. Note this skill is different from the existing skill `Tr_Fl`.
- `Pi_Bx` and `Tr_Bx` to pick and translate box, respectively. They are demonstrated in a similar way to `Pi_St` and `Tr_St` for alphabet “R”.

The number of demonstrations and training time for these skills are similar to their counterparts without box. The learned skill models of `Re_St2Fl_Bx` and `Tr_St_Bx` are shown in Fig. 6.

The desired behavior of this task is summarized below:

- If the initial and goal orientations of the alphabet are the same (i.e., both standing or lying flat), then the desired sequence is the same as specified in Sec. II-A3. Namely,
  - when the alphabet is initially standing, the desired skill sequence is `Pi_St` → `Ro_St` → `Tr_St` → `Reset`
    or `Pi_St` → `Tr_St` → `Ro_St` → `Reset`.
  - when the alphabet is initially lying flat, the desired skill sequence is `Pi_Fl` → `Ro_Fl` → `Tr_Fl` → `Reset`
    or `Pi_Fl` → `Tr_Fl` → `Ro_Fl` → `Reset`.
- However, if they are different (i.e., from standing to lying-flat or vice versa), then the robot should re-orient the alphabet from standing to lying-flat or vice versa, by using the levitation box (instead of the edge of the platform). Note that as mentioned earlier, these re-orientation skills `Re_St2Fl_Bx` and `Re_Fl2St_Bx` allow the alphabet to be *directly* re-oriented from the platform surface to the box top without the need for re-grasping. Moreover, in order to use the box, the box should be translated from its original pose to around the middle of the platform to facilitate the re-orientation (the exact pose depends on the model of these re-orientation skills). More specifically,
  - when the alphabet is initially standing and lying-flat as the goal, the desired skill sequence is `Pi_Bx` → `Tr_Bx` → `Reset` → `Pi_St` → `Re_St2Fl_Bx` → `Reset` → `Pi_Fl` → `Ro_Fl` → `Tr_Fl_Bx` → `Reset` → `Pi_Bx` → `Tr_Bx` → `Reset`;
    or `Pi_Bx` → `Tr_Bx` → `Reset` → `Pi_St` → `Re_St2Fl_Bx` → `Reset` → `Pi_Fl` → `Tr_Fl_Bx` → `Ro_Fl` → `Reset` → `Pi_Bx` → `Tr_Bx` → `Reset`.
  - when the alphabet is initially lying-flat and standing as goal, the desired skill sequence is `Pi_Bx` → `Tr_Bx` → `Reset` → `Pi_Fl` → `Re_Fl2St_Bx` → `Reset` → `Pi_St` → `Ro_St` → `Tr_St_Bx` → `Reset` → `Pi_Bx` → `Tr_Bx` → `Reset`;
    or `Pi_Bx` → `Tr_Bx` → `Reset` → `Pi_Fl` → `Re_Fl2St_Bx` → `Reset` → `Pi_St` → `Tr_St_Bx` → `Ro_St` → `Reset` → `Pi_Bx` →

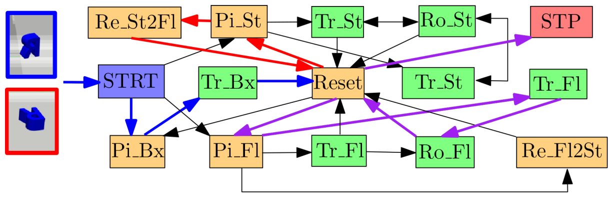

Fig. 7. The learned GTN associated with **Task-2** described in Sec. II-A4, which has 15 nodes and 27 edges (with 32 embedded TPGMMs).

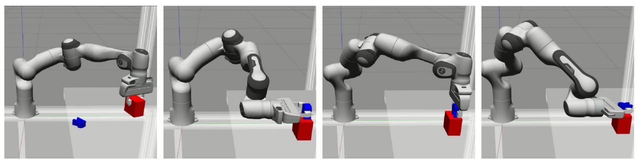

Fig. 8. Snapshots of execution for one case of **Task-2** in Sec. II-A4.

`Tr_Bx` $\rightarrow$ `Reset`.

As a result, **Task-2** requires 4 possible discrete sequences with length 4, 4, 14 or 14, of which contains 2, 2, 4 and 4 6D skill parameters (*each* as 6D poses). Snapshots of the execution of one case are shown in Fig. 8.

We generated 150 problems of **Task-2** for training, 40 for each case above. The problems are generated by randomly sampling from possible poses of the alphabet as initial and goal poses, with a equal distribution over the above four different cases. Note that the initial and final positions of the box are always at $(-0.1, 0.7)$, i.e., there is no benefit of moving the box judging from the goal. Then, these problems are solved by the hybrid TAMP described in Sec. II-A2. It took in total $18\,\mathrm{h}$ (in average 6 minutes per problem) to generate 150 successful plans as in (9), using the exhaustive hybrid TAMP mentioned above. As described in the learning algorithm of GTN, the associated GTN is learned in $7.8\,\mathrm{s}$ with 15 nodes and 27 edges (with 32 embedded TPGMMs), as shown in Fig. 7. Its topology matches the anticipated plans described above well. In particular, the robot can manipulate the alphabet directly or pick and translate the box to the center of the platform if re-orientation of the alphabet is needed. Afterwards, the box is then picked and translated *back* to its original pose. The learned GTN is used to solve another 150 problems in the validation set by following the execution procedure. Similar as before, starting from the initial state, the learned GTN can directly output the best next skill and the associated skill parameters, based on the current state and the goal state. As a result, only the states and skills on the optimal path are explored and thus significantly reduce the planning time. More specifically, it took in total $15\,\mathrm{min}$ (in average $8.9\,\mathrm{s}$ per problem) to solve all problems in the validation set with a success rate of 96%, resulting in a 100-fold decrease in planning time while maintaining the same success rate as the training data.

5) Task-3: Learning and Execution Results: All skills necessary for this task are already introduced in **Task-2** for the box and alphabet "R", whereas additional skills for the new alphabets "F" and "G" can be re-demonstrated, re-learned or modified from those skills of "R".

Since now there are in total 4 objects, the desired behavior of this task is quite complex and summarized below:

- If the goal configuration does *not* require the alphabets to be stacked, each alphabet can be manipulated to the goal state in arbitrary order, each of which follows the same procedure as described in Sec. II-A5 for **Task-2**. One subtle difference is that the box only needs to be moved once if more than one alphabets require re-orientation.
- If two or three alphabets are stacked while being lying-flat in the goal configuration, the alphabet at the *bottom* of the stack should be manipulated first, then towards the *top* of the stack. This is because alphabets on the higher level can not be stacked without the lower alphabets being stacked first. Consider the goal configuration of all three alphabets in one stack. There are in total 6 possible arrangement, each leading to a different sequence of manipulating the alphabets. For instance, if the goal stack is "R, F, G" from bottom to top, then the sequence of manipulation is "R" first, "F" second, and "G" last.
  On top of that, given different initial orientations of the alphabets, re-orientation with the levitation box should be used before translation and rotation towards the goal state. Same to the previous case, the box only needs to be moved once if more than one alphabets require re-orientation.
  Note that the cases where the alphabets are stacked while being standing are not considered here, due to the frequent fall of such stacked structure.

There are in total 13 different skill sequences under 18 different cases, of which the longest plan has 27 skills and the shortest plan has 12 skills (with 8 and 6 6D parameters, respectively). Snapshots of the execution of one case are shown in Fig. 10.

Due to the extremely long sequence of the solutions, the above problem is much *more* difficult thus takes much longer for the classic TAMPs to find the solution, no matter whether they are combined with LfD skills or motion planners, For training, 500 problems are created (40 problems for each case above). More specifically, the solution time increases drastically by 50-fold, as it in average takes $4\,\mathrm{min}$ to solve tasks that do not require re-orientations of the alphabets, while $8\,\mathrm{min}$ to solve tasks that require re-orientations of the alphabets. In particular, there are 29 potential skills and 14 6D parameters to expand at *each* state of the graph search. Moreover, an incorrect sequence of manipulating the alphabets would yield large part of the search tree invalid, thus expensive to explore. Given the training data, it took $4\,\mathrm{min}$ to learn the associated GTN which has 29 nodes and 53 edges (with 256 embedded TPGMM components). The large number of TPGMM components is due to the large number of all possible unique plans as mentioned above.

The learned GTN is shown in Fig. 9. Its structure and embeddings correctly model the following constraints: (I) the box is used only when at least one of the alphabets needs re-orientation, i.e., the transition from skill `Reset` to skill `Pi_Bx`. (II) the alphabets are manipulated based on their levels in the goal stack, i.e., the alphabet on the bottom is manipulated first, i.e., the transitions from skill `Reset` to skills `Pi_St_R`, `Pi_St_F`, `Pi_St_G`, `Pi_Fl_R`, `Pi_Fl_F` and `Pi_Fl_G`. The learned GTN is used to solve another 520 new

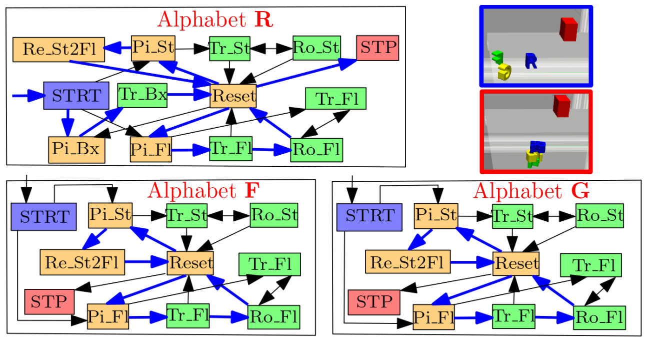

Fig. 9. The learned GTN associated with **Task-3** described in Sec. II-A5, which has 29 nodes and 53 edges (with 256 embedded TPGMMs). Note that for the ease of visualization, the skill `Reset` is duplicated across three alphabets.

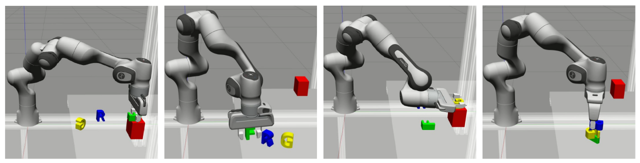

Fig. 10. Snapshots of execution for different cases of **Task-3** in Sec. II-A5.

problems in the validation set. The edges of learned GTN in average take longer to evaluate due to increase number of components. It took in total 2.9 h (in average 20 s per problem) and a success rate of 95%, which still results in a more than 100-fold decrease in planning time. It is worth mentioning that due to the large number of components within the embeddings, it took around 1.5 s to choose each skill along the plan, including optimizing the task parameters and maximizing the observation probability.

B. Industrial Assembly on Hardware

Last but not least, we consider parts of an industrial assembly task on hardware as shown in Fig. 12. The actual Panda robot is used within a workspace that consists of a feeding and inspection platform; and an assembly station where various pieces are assembled into a product. The platform is monitored by a Zivid 3D camera from [12], from which the collected point-clouds are inputs to the point-pair-feature detection algorithm see [13]. It provides a 6D pose estimation with around 1 cm accuracy w.r.t. the global frame. Moreover, a task-space impedance controller as proposed in [14] is used to track Cartesian reference trajectories.

Similar to **Task-2**, there are two cases associated to this part of the assembly process: **Case-1**: pick a non-defective metal cap from the platform, and attach it to the top of a metal peg on the assembly station; **Case-2**: pick a defective metal cap from the platform, and drop it to a pallet. Both the cap and the peg are components of the e-bike motor.

1) Primitive Skills: As summarized in Table III, we demonstrated in total 10 skills relevant to the task directly via kinesthetic teaching on the robot. The state of the end-effector and the gripper are fetched directly from the on-board control manager, while the poses of the objects (e.g., the cap and the peg) are retrieved from the perception system. Demonstrations are recorded at 50 Hz. In addition to the skills described in

Skill Name	M_a	K_a	TP_a	$t\,(\theta_a\,\vert\,\gamma_a)$ [s]
`In_Pg`	8	8	$\{P, R\}$	3.2 | 4.2
`Pi_St`	12	7	$\{C, R\}$	3.6 | 2.7
`Pi_Si`	10	8	$\{C, R\}$	4.2 | 3.1
`Pi_Fl`	8	7	$\{C, R\}$	3.8 | 3.2
`Re_Fl2St`	10	8	$\{C, R, C_G\}$	3.8 | 2.2
`Ro_St`	15	6	$\{C, R, C_G\}$	4.2 | 2.7
`Tr_St`	10	7	$\{C, R, C_G\}$	4.1 | 1.9
`At_St`	10	7	$\{C, R, C_G\}$	4.3 | 2.1
`Dr_Fl`	10	6	$\{C, R, C_G\}$	3.3 | 2.2
`Reset`	7	6	$\{R\}$	2.8 | 1.5

TABLE III
DEMONSTRATED SKILLS C, NUMBER OF DEMONSTRATIONS M_a, NUMBER OF COMPONENTS K_a, CHOICE OF TASK PARAMETERS TP_a, AND TRAINING TIME FOR θ_a AND γ_a. REGARDING FRAMES, "C, P, R, C_G" STAND FOR THE INITIAL POSE OF CAP, THE INITIAL POSE OF PEG, THE INITIAL POSE OF THE ROBOT ARM AND THE DESIRED CAP GOAL, RESPECTIVELY.

Task-1 of Sec. II-A1 for the alphabets, the following three skills are added:

- `In_Pg`: to insert the peg into the workstation.
- `At_St`: to attach the cap onto the top of the peg while standing. Due to limitations of the perception system, the positions of the peg and the workstation are not changed across demonstrations.
- `Dr_Fl`: to drop the cap into the container while lying-flat. Due to limitations of the perception system, the positions of the cap and the pallet are not changed across demonstrations.

2) Assembly Task: Learning and Execution Results: First of all, the skill model $\mathcal{M}_a$ is learned for each skill, of which the details such as the training time and the choice of frames are given in Table III. On average, 10 demonstrations are needed for each skill and the model is learned within 7 s.

With these skills, the aforementioned two cases can be addressed by the following.

- **Case-1**: If the cap is initially standing-up, the robot should insert the peg first into the workstation. Then it should pick up the cap and attach it onto the peg, i.e., `In_Pg`→ `Reset`→ `Pi_St`→ `At_St`→ `Reset`. However, if the cap is initially lying flat, after picking the cap, the robot should re-orient it to standing position. Then the robot can pick it up and attach it onto the peg. i.e., `In_Pg`→ `Reset`→ `Pi_Fl`→ `Re_Fl2St`→ `Reset` → `Pi_St`→ `At_St`→ `Reset`.
- **Case-2**: If the cap is initially lying flat, the robot should directly pick up the cap and drop it into the container, i.e., without inserting the peg, i.e., `Pi_Fl`→ `Dr_Fl`→ `Reset`. However, if the cap initially standing-up, the robot should rotate and translate it first to the platform edge, and then pick it up from the side before dropping it into the container, i.e., `Pi_St`→ `Ro_St`→ `Tr_St`→ `Reset`→ `Pi_Si`→ `Dr_Fl`→ `Reset`.

Note that the skills `At_St`, `Dr_Fl`, `Ro_St` and `Tr_St` all have one free skill parameter as 6D pose. Clearly, the desired skill sequence for different cases above depends on the initial

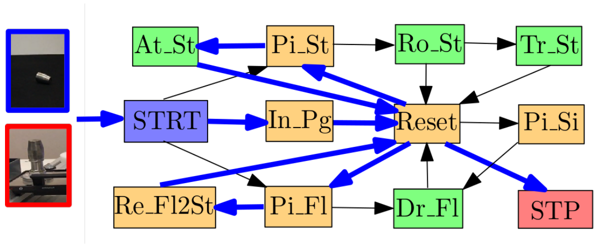

Fig. 11. The learned GTN associated with **Assembly-Task**, which has 12 nodes and 19 edges (with 27 embedded TPGMMs).

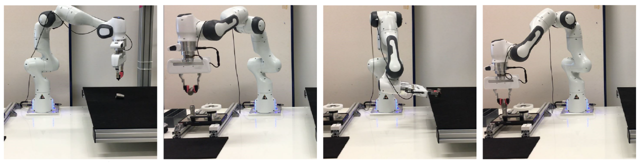

Fig. 12. Snapshots of execution of the **Assembly Task** in Sec. II-B: left-two for placement and right-two for insertion.

system state, particularly, the initial pose of the cap. For **Case-1**, since the skill `At_St` 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 **Case-2**, since the skill `Dr_Fl` is only allowed when the cap is grasped from the side or flat (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. Similar to **Task-1**, the re-orientation and grasping of the cap from the side are only taught at the edge of the platform to avoid collision between the robot arm and the platform surface.

We generated 80 problems of **Assembly-task** for training, within which the two cases described above are covered evenly. The problems are generated by randomly sampling from possible initial poses/orientations of the cap on the platform surface and different goal states for insertion or dropping. Then, these problems are solved by the hybrid TAMP described in Sec. II-A2. It took in total 2.5 h (in average 6 minutes per problem) to generate 80 successful plans as in (9), using the exhaustive hybrid TAMP mentioned above. As described in the learning algorithm of GTN, the associated GTN is learned in 1.3 s with 12 nodes and 19 edges (with 27 embedded TPGMMs). The learned GTN topology as shown in Fig. 11 matches the anticipated cases above well. For instance, the peg will not be inserted if the goal is to drop the cap; and the cap is re-oriented from lying flat to standing to be inserted. Another 80 problems are created for validation, it took in average 3.8 s to solve one problem with a success rate of 100%, resulting in a 100-fold decrease in planning time. Furthermore, the learned GTN is also used to control the robot during online execution. Namely, given the goal state from the problem definition and the initial state from perception, the learned GTN is used to directly output the best next skill and the associated skill parameter. Note that the generated reference trajectory in the task space for each skill is sent to the task-space impedance controller which runs at 1 kHz.

3) Details on Baselines: We compare our approach against the following four baselines:

- (**Hb-lfd**): the hybrid TAMP planner combined with the skill representations learned from LfD using the confidence measure proposed in [6] as the searching heuristics. More specifically, for each edge, the confidence measure described in (7) is used to *estimate* the cost of an edge in the TAMP search graph, i.e., the edge cost is small when its confidence measure is high. In other words, the tree expansion is biased towards skills that are more confidence at the current state.
- (**Hb-mp**): the hybrid TAMP planner combined with skills generated from sampling-based motion planner RRTC see [10] with no heuristics. Different from (**Hb-lfd**), the system evolution at each state is evaluated by executing each skill *within* the simulator, by the proposed motion planner. Moreover, the edge cost is obtained also from the simulator as the control cost or execution time. The simple "distance to goal" is used as the search heuristic. Same setup as the previous planner, but now the edge cost is fused with another measure which is simply the Euclidean distance between the current state and the goal state. In other words, the skill that drives the system closer to the goal state is preferred during the search expansion.
- (**MLP**): a multilayer feedforward neural network to approximate the choice of both discrete and continuous parameters. A 3-layer structure is used where the input includes *both* the system state and the goal state, whether the optimal choice of next skill and the parameters are used as the output. The hidden layer has 512 tensors, and the learning rate for the Adam optimizer is set to 1×10^{-3} The loss function is the MSE between the target choice of skill and parameters and the predicted one. A batch size of 16 is used for the training. During evaluation, we split the error in predicting the skill parameters and the choice of the skill.
- (**NPI**): a LSTM-based policy network similar to the Neural Programmer Interpreter from [15]. It is proposed to train a network to imitate a computer program, i.e., start or stop programs recursively given certain context. We use one hidden layer with a memory of 5 steps. The output and the loss function remain the same as used in **MLP**, but the input is now a *sequence* of past system state, goal state and the chosen skill. The hidden layer has dimension 500, and the batch size is set to 32.

The metrics to compare are the solution time in the validation set, and the success rate when the planning *time limit* is set to 100 times the solution time of the GTN planner.

References

[1] 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.

[2] 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.

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

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

[5] 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.

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

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

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

[9] E. Franka, "Panda arm," https://www.franka.de/panda/, 2018.

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

[11] M. Spies, M. Todescato, H. Becker, P. Kesper, N. Waniek, and M. Guo, "Bounded suboptimal search with learned heuristics for multi-agent systems," in *AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 2387–2394.

[12] Ø. Skotheim, H. Schumann-Olsen *et al.*, "Zividlabs," *Zivid 3d camera*, 2020.

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

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

[15] S. Reed and N. de Freitas, "Neural programmer-interpreters," in *International Conference on Learning Representations (ICLR)*, 2016.