2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Enabling Maintainablity of Robot Programs in Assembly by Extracting Compositions of Force- and Position-Based Robot Skills from Learning-from-Demonstration Models

Daniel Bargmann[1], Werner Kraus[1] and Marco F. Huber[1,2]

***Abstract*— To this day, only a small number of industrial robots is used in assembly. One key reason for this is that specific contact situations require the introduction of force-control schemes. The parameters for those schemes are hard to select in practice, because they require in-depth expertise about the robot and the process. Learning-from-Demonstration (LfD) provides a powerful approach to intuitively parameterize robot programs by demonstrating the task at hand. However, when dimensions increase by including force or orientation, many LfD algorithms are hard to verify, understand and maintain, requiring expert knowledge to make adaptions, effectively making it a "black-box". This property renders them ineffective for usage in industrial applications. We build upon a system of composable *skills*, that can be easily adapted by experts without the need to demonstrate the task again. This approach to *skill*-based robot programming promises to address the issues of readability and maintainability by sequencing robot movements in *skills* and breaking them down into understandable (sub-)goals. In this paper, we combine skill-based programming with LfD, preserving both maintainability and intuitive parameterization. We present (a) an approach to parameterize and create sequences of hierarchies of force- and/or position-controlled robot *skills* from a LfD model, (b) which can be adapted by a user by hand with few, basic and understandable parameters, and (c) show its applicability on the real-world example of terminal clamp assembly. We achieve a reduction in teach-in time of 53.8% for variants, increased robustness against variance, and efficient tight stacking of clamps with a gap of ≤ 1 mm.**

I. Introduction

Only about 6 % of robotic operational stock in North America and Europe are deployed for assembly tasks, a prominent reason for that is the time for setting up and programming robots [1]. This requires robot and process experts, which are both expensive and scarce. There have been many approaches to make their work more efficient or reduce the required knowledge. One of those approaches is Learning-from-Demonstration (LfD), where operators demonstrate the process and the robot learns a suitable program. This is especially beneficial for teaching force-based robot programs, which are generally harder to program than pick-and-place programs. There are some particular challenges that interfere with the application of LfD to productive use-cases. LfD programs based on trajectories are hard to understand, especially when the number of dimensions increases. A typical force-based robot program trajectory requires about nine to twelve dimensions (three for position, orientation, forces, and torques, respectively). Such complexity typically cannot be evaluated by non-experts, thus assessing the quality or correctness of the program becomes challenging. Furthermore, LfD programs are often monolithic, i.e., they cannot be easily adapted to new situations, e.g., new product variants. In the manufacturing industry, reliability and maintainability are key for the adaption of a system. The program needs to be interpreted and adapted by experts quickly. The lack of these features in many approaches might be the reason for their low adaption rate in productive systems. Skill-based methods promise to address these issues by defining sub-programs (*skills*) that aim to be adaptable, reusable, and human-readable [2]–[4]. The main benefit of these *skills* is a human-readable structure that reduces the complexity of the robot programs. However, these *skills* still need to be parameterized and arranged.

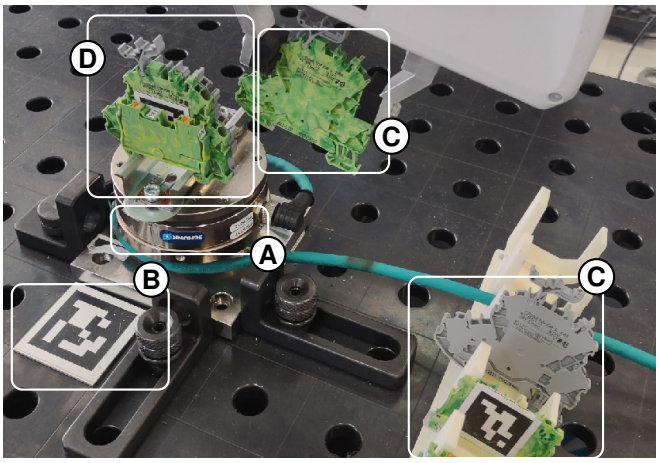

Fig. 1: (A) Assembly rig with a DIN rail mounted onto a force-torque sensor, (B) reference marker, (C) terminal clamps and (D) tightly stacked terminal clamps

Typically there are several different levels at which operators and programmers can manipulate robot programs, with different levels of abstractions. Albus et al. [5] introduced a hierarchy of task levels at which a robot program can be manipulated. Although it was originally intended for

[1]The authors are with the Fraunhofer Institute for Manufacturing Engineering and Automation, Stuttgart, Germany {danb, wek, mrh}@ipa.fraunhofer.de

[2]Institute of Industrial Manufacturing and Management IFF, University of Stuttgart, Germany

This work was supported by the Baden-Wuerttemberg Ministry for Economic Affairs, Labour and Tourism under *AI Innovation Center (Learning Systems and Cognitive Robotics)* – Grant No. 017-180036 and the German Federal Ministry for the Environment, Nature Conservation, Nuclear Safety and Consumer Protection under *Desire4Electronics (KI Leuchttürme)* – Grant No. 67KI31054A.

teleoperated robots, this taxonomy of interaction levels stays valid also for the application of non-teleoperated robots. 6 levels are introduced: Level 1 Servo (i.e., joint level), Level 2: Primitive Motions (i.e., trajectories), Level 3: E-Moves (e.g., intermediate movements), Level 4: Task Level (i.e., a series of e-moves, performed on manipulated objects) and Level 5 and 6, which define higher levels of abstractions.

Typically a lower level of interaction (e.g., joint level) enables user to manipulate the resulting motions/program more precisely, but on the other hand requires more time and expertise to understand and maybe adapt the motion. Typically, many LfD approaches work either on Level 1 and 2 (joint values, trajectories) or directly parameterize tasks (Level 4), and abstract the level of intermediate movements.

In this paper, we show that we can generate such intermediate movements (Level 3) from lower level representations learned by traditional LfD algorithms. We especially focus of force-based robot programs and show that such representation can be beneficial for experts that need to interpret, adapt (e.g., for new requirements) and maintain (e.g., in the case of errors) such force based robot programs. We achieve this by

1) introducing a method to sequence, compose and parameterize force-based predefined *skills* directly from a trajectory-based LfD model and
2) show that this parameterized robot program can be a) easily adapted to new product variants (*adaptability*) b) extended with expert knowledge to a different domain/requirements (*interpretability* and *maintainablity*)

We validate our approach on the following two use-cases: A real-world industry-relevant use-case of terminal clamp assembly and a 2D letter drawing dataset. To our knowledge, there is no approach that segments LfD and trajectory-based force-controlled robot programs into a sequence of such composable *skills*.

We specifically do not target non-experts, but focus specifically on experts. The reason for this approach is that in current industry settings, experts have to evaluate and maintain the robot program, thus experts need the ability to change those programs in a fine grained manner. Additionally, we do not want to thoroughly proof the usability of our chosen representation, as this representation has been established in previous work. Our evaluation should show that the switch in representation enables a manipulation of the robot program, that was previously exceedingly hard.

II. Related Work

After being introduced by Hasegawa et al. [6], *skill-based* robot programming has been studied extensively. Commonly, many approaches build upon the *Task Frame Formalism* [7]. In this approach, a hybrid position and velocity based control scheme is applied to a *task frame*, a specific frame defined in relation to the controlled robot. A number of publications extend this approach with their own coordination mechanisms and Domain Specific Languages (DSLs) [8], [9], which are designed to be human-readable and maintainable.

As mentioned before, Albus et al. [5] provide a hierarchical classification of interaction levels, ranging from joint motions to high level task representations (NASREM).

The specific approach that this paper will build upon is *pitasc* defined by Naegele and Halt [2], [3] based on the *iTaSC* algorithm [10]. Here, *skills* define robotic capabilities such as, linear motions, force control or arbitrary combinations of these (see [11]). One distinctive feature of this approach is the definition of *skills* by composing several, low level *skills* or controllers to build more complex behaviors, which corresponds to level 3 and 4 in the NASREM hierarchy level. This helps users to break down complex movements into understandable parts, which can be adapted more easily. The behavior of a *skill* is fixed, not learned and therefore, the behavior is clear to the user as it does not change when employed beyond the original domain. One notable exception is the *semantic skill recognizer* by Steinmetz et al. [4], where predefined *skills* are segmented from a demonstrated processes. Adapting those *skills* is still a high effort, as they are designed to be operated on task level (i.e., level 4). Recent advances [12] try to learn and discover *skills* automatically and to increase reusablity by applying *skill* parameterization, meta-learning, or adapting to new domains. There are works (e.g., Pastor et al. [13]), that provide adaptability for *skills* but still miss a way to interpret those *skills*, as they only provide a high level interface to skill adaption (level 4). Trajectory-based LfD algorithms include Locally Weighted Regression (LWR) [14], Gaussian-Mixture-Models (GMMs) [15], Hidden-Markov-Models (HMMs) with multivariate Gaussian kernels [16], and Probabalistic Motion Primitives (ProMPs) [17]. These either model uncertainties directly [15]–[17] or use regression to minimize errors [14]. However, these models are limited in their adaptability and interpretability. Skills are defined as a set of Motion Primitives (MPs), which are hard to interpret, especially with forces and orientations. To tackle the problem of reusability, Calinon et al. introduced Task-Parameterized (TP) GMMs [15], where several trajectory models are encoded in different coordinate frames (e.g., robot end-effector), each are parameterized by the position and orientation of their respective coordinate frame. Later, they are re-parameterized with new coordinate frames. All models are fused by combining the respective Gaussians, yielding one new model, conditioned on the new coordinate frames. There have been several works trying to combine constraint-based algorithms, specifically the *iTaSC* algorithm, with LfD approaches. Perico et al. [18] combined learned trajectories with constraints defined by experts. This approach allows adapting the trajectories based on the variability of the taught algorithms while respecting expert knowledge such as a specific force-controlled behavior. Silverio et al. [19] learned task prioritization hierarchies from demonstration by matching them to a set of candidate hierarchies, that are evaluated during runtime. Those approaches mostly provide an interface on hierarchy level 2. In the field of Reinforcement Learning (RL), a few researchers investigated learning structured robot programs by parameterizing them

TABLE I: Symbols and Descriptions

Symbol	Description	Symbol	Description
Problem Description			
$\mathcal{S}$	Skill	n	Number of Skills
$\mathcal{H}$	Hierarchy Level	m	Number of Subgoals
$\mathcal{C}$	Controller	$\mathbb{V}$	Spacial Description
$\mathcal{P}$	Parameter set	$\mathcal{D}$	Dataset of Demos $\mathcal{K}_i$
$\mathcal{K}$	Single Demonstration	$\mathcal{T}$	Terminal Condition
$\xi_t^{(i)}$	Datapoint of Demo $\mathcal{K}_i$ at time t	$\mathcal{M}$	Intermediate Representation
TP-HSMM $\mathcal{M}$			
$\alpha_{i,j}$	transition probability from state i to j	K	Number of States
T_j	Transformation from coordinate frame (j)	c_j	offset from coordinate frame (j)
$\mu_i^{(j)}$	mean w.r.t. coordinate frame (j)	$\Sigma_i^{(j)}$	covariance matrix w.r.t. coordinate frame (j)
μ_i^D	mean of lognormal of state duration	Σ_i^D	covariance matrix of lognormal of state duration
Process Parameters			
$F_{x,y,z}$	Forces in x, y, z axis, respectively	K_{adm}	Admittance Gain

[20], [21]. Sharma et. al. [22] used RL to learn a priority ordering or hierarchy of controllers based upon predefined axes and controllers in simulation, which are based upon the workpiece geometry or handpicked. One disadvantage of RL methods is their setup time. Usually, a simulation is required, which is often inaccurate for in-contact situations such as assembly. To reduce these inaccuracies of the simulation (*sim2real gap*) additional efforts are needed, e.g., manual tweaking of parameters, techniques such as *domain randomization* or model verification. In conclusion, LfD and other learned approaches lack interpretability and are hard to adapt manually. *Skill-based* approaches on the other hand lack the easy teach-in of LfD approaches. Approaches either provide interfaces on a low level (level 1/2) or on a higher level (level 4). Combining LfD, and a compositional, *skill-based* approach to defining skills, which exposes a level 3 interface, can thus introduce interpretability, especially for force-based robot programs. Such a unified approach is missing in the literature.

III. Problem Description

We define a robot program that is interpretable and maintainable by these attributes:

1) It can be broken down into elements, each with a single goal per axis,
2) has a clearly defined sequence of actions, where
3) each action has one/several clear terminal condition and
4) exposes only a limited set of parameters to the user.
5) Each action is easy to extend and modify and
6) newly added behavior can be added easily by overwriting existing one (i.e., if a behavior is added that conflicts with previous behavior, the new one takes precedence).

The structure of such a problem can be formulated as such:

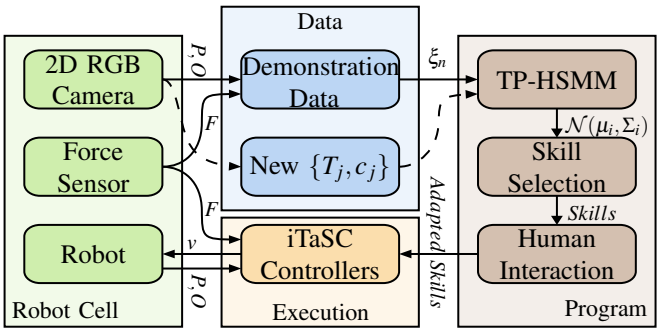

Fig. 2: Overall structure of the control and learning approach. Humans can adapt the created program in the last step before it is executed by the iTaSC controller.

$$\mathcal{RP} = \{(\mathcal{S},\mathcal{T})_1, ..., (\mathcal{S},\mathcal{T})_n\} \ , \tag{1}$$

$$\mathcal{S} = \{(\mathcal{C},\mathcal{P},\mathcal{H},\mathbb{V})_1, ..., (\mathcal{C},\mathcal{P},\mathcal{H},\mathbb{V})_m\} \ , \tag{2}$$

where $\mathcal{RP}$ denotes the robot program, defined by a set of n elements, each with (multiple) terminal conditions $\mathcal{T}$ and a *skill* $\mathcal{S}$, which consist of m goals, denoted by a controller $\mathcal{C}$, the parameters $\mathcal{P}$, a hierarchy level $\mathcal{H}$, and a spacial description $\mathbb{V}$ (e.g., a coordinate system). On the other hand, a dataset of demonstrations $\mathcal{D}$, which is acquired in a LfD setting is described by $\{\mathcal{K}_1, ..., \mathcal{K}_o\}$ demonstrations, where $\mathcal{K}_i = \{\xi_t^{(i)}\}_{t=1}^N$, $\xi_t^{(i)} \in \mathbb{R}^d$ with d being the dimension of a datapoint $\xi_t^{(i)}$. To convert a set of demonstrations $\mathcal{D}$ to a robot program $\mathcal{RP}$, (a) $\{\mathcal{K}_1, ..., \mathcal{K}_o\}$ demonstrations have to be combined into a single representation $\mathcal{M}$, which (b) has to be split into n *skills* $\mathcal{S}$ and (c) a list of accompanying terminal conditions $\mathcal{T}$ has to be selected. (d) Each *skill* $\mathcal{S}_i$ has to be separated into m subskills, each with a single goal, (e) which has to be parameterized with a minimal set of parameters $\mathcal{P}$. A table overview over all symbols can be find in Tab. I.

IV. Methods

In this work, without loss of generality, we chose the *pitasc* skill system of Naegele et al. [2], [3], [11] for skill representation and execution, as it implements a system compliant to Eq. 1, 2. To further improve interpretability, these *skills* can also be easily visualized by a GUI or by a Mixed-Reality device (see Krieglstein et. al [23]). As intermediate representation $\mathcal{M}$, we chose Task-Parameterized (TP)-Hidden-Semi-Markov-Model (HSMM), which provides representation and a segmentation into *skills* $\mathcal{S}_i$ based on Motion Primitives (MPs). The TP-HSMM is learned from demonstrated trajectories, which is then adapted to the current workpiece and part positions of the task (see Sec. IV-A). Based on this, we propose a novel approach to deduce a set of terminal conditions $\mathcal{T}_i$, where we consider force, duration, and position (Sec. IV-C) and a segmentation of each *skill* $\mathcal{S}_i$ into a set of single goals described by $(\mathcal{C},\mathcal{P},\mathcal{H},\mathbb{V})_i$ (Sec. IV-B). To achieve this, we calculate a coordinate frame for each *skill* $\mathcal{S}_i$. For each axis of the coordinate frame, we estimate an admittance gain. Axes with an admittance gain within a plausible window are accepted as force controlled axes,

others are considered position-based (Sec. IV-B.2). The parameters for each skill are extracted from the MPs definitions (Sec. IV-B). The complete *skill* acquisition pipeline is shown in Fig. 3 and is embedded in the overall system (Fig. 2).

A. Model $\mathcal{M}$: Task-Parametrized Semi-Markov Models

As intermediate model, we encode our demonstrations into a TP-HSMM, which is an extension of a traditional HMM for motion encoding. The main reason for selecting this approach is the data-efficiency of such methods, i.e., teaching is possible in a reasonable amount of time, without simulation or specialized equipment.

Additionally it already provides a segmentation in form of MPs as states of a Markov-Chain. The parameters for a HSMM with K states and a multivariate Gaussian as output distribution can be denoted as

$$\{\{\alpha_{i,j}\}_{j=1}^{K},\ \Pi_i,\ \mu_i,\ \Sigma_i,\ \mu_i^D,\ \Sigma_i^D\}_{i=1}^{K}\ , \tag{3}$$

where $\alpha_{i,j}$ is the transition probability from the hidden process state i to j, Π is the mixing coefficient and μ, Σ the mean and covariance matrix for the respective Gaussian. μ_i^D, Σ_i^D are the parameters of a lognormal distribution that models the state duration.

Task-Parameterization [15] extends this approach by augmenting the dataset and states the orientation (as transformation matrix T_j) and offset (as vector c_j) of different coordinate frames j (e.g., the coordinate frame of the workpiece). The learned parameters μ_i, Σ_i for each coordinate frame can be adapted for a new coordinate frame by

$$\hat{\mu}_i^{(j)} = T_j\mu_i^{(j)} + c_j\ ,\ \hat{\Sigma}_i^{(j)} = T_j\Sigma_i^{(j)}T_j^{\mathrm{T}}\ , \tag{4}$$

with the $\hat{\mu}_i^{(j)}$ and $\hat{\Sigma}_i^{(j)}$ as representation of the re-parameterized model for the coordinate frame j, respectively.

These multiple Gaussians can be combined into a single one representing the same HSMM state, due to the linear transformation property of normal distributions, by

$$\hat{\Sigma}_i = (\sum_{j=1}^{P}(\hat{\Sigma}_i^{(j)})^{-1})^{-1}\ ,\ \hat{\mu}_i = \hat{\Sigma}_i\sum_{j=1}^{P}(\hat{\Sigma}_i^{(j)})^{-1}\hat{\mu}_i^{(j)}\ . \tag{5}$$

This can be repeated for all states, resulting in a combined, *collapsed* HSMM. We consider MPs that are reached from the current start state as skills $\mathcal{S}_i$, which results in a segmentation.

B. Atomic Goals

Each MP representing a $\mathcal{S}_i$ now needs to be split into atomic goals $(\mathcal{C},\mathcal{P},\mathcal{H},\mathbb{V})_i$. Our overall approach can be seen in Fig. 3, where blue shaded parts represent novel steps. Naturally, three fundamental types of explicit goals arise: position, velocity and force. Here, we consider force and position, into which the MP need to be broken down to. For this, we first define a spacial representation $\mathcal{V}$ which is an axis and a corresponding coordinate frame in our case.

1) Coordinate Frames: With Sec. IV-A, a *skill* $\mathcal{S}$ is defined by the Gaussian

$$\mathcal{N}(\xi_i;\hat{\mu}_i,\hat{\Sigma}_i)\ , \tag{6}$$

$$\text{with } \xi_i = [x,y,z,v_x,v_y,v_z,q_w,q_x,q_y,q_z,F_x,F_y,F_z]\ , \tag{7}$$

where x, y, z denote the position, q_w, q_x, q_y, q_z the orientation in unit quaternion representation, F_x, F_y, F_z the Cartesian force measurements and v_x, v_y, v_z the velocities.

We can find a coordinate frame for the most distinct goals of a corresponding *skill* $\mathcal{S}_i$ by applying an eigenvalue decomposition of the covariance matrix $\hat{\Sigma}_i$ by

$$\hat{\Sigma}_i = VDV^{\mathrm{T}} = \sum_{j=1}^{m}\lambda_j v_j v_j^{\mathrm{T}}\ , \tag{8}$$

where $V = [v_1,v_2,...,v_m]$ and $D = \mathrm{diag}(\lambda_1,\lambda_2,...,\lambda_m)$ are the eigenvectors and the matrix of the eigenvalues, respectively, with m being the number of dimensions of the covariance matrix. Typically, the eigenvector v_s with the smallest eigenvalue λ_s corresponds to the axis, where the data has the least variance, thus the most distinct movement (minimal intervention principle [24]).

Our assumption is that, whenever forces need to be applied (i.e., in contact cases) a description defined with an explicit force goal is most sensible and position goals should only be considered if a force goal is not sensible. This means that the coordinate frame of a skill's spacial description will be dominated by forces. It is thus sufficient to only consider the projection of $\mathcal{N}(\xi_i;\hat{\mu}_i,\hat{\Sigma}_i)$ into the force domain. To achieve this, we apply the eigenvalue decomposition to the projection of $\hat{\Sigma}_i$ into the force domain, which we denote by $\hat{\Sigma}_{F_i}$.

We receive a new coordinate frame, which is oriented to the most distinct axes regarding force goals, by sorting the eigenvectors based on the eigenvalues. We order the eigenvectors by

$$\mathbb{V}_{S_i} = [v_s,...,v_j]^T,\ \text{with } \lambda_s < ... < \lambda_j\ , \tag{9}$$

which creates a coordinate frame $\mathbb{V}_{S_i}$ where the axes are ordered by their respective force goals' importance. The goal value of each axis is extracted from the mean values of the MPs projected into the new coordinate frame by $\mu_g = \mathbb{V}_{S_i}\hat{\mu}_i$. This is done explicitly for the force part of $\hat{\mu}_i$ and implicitly for the position part (see IV-D). Additionally, this results in a strict hierarchy level $\mathcal{H}$ of each goal.

2) Controller type $\mathcal{C}$: An axis in the coordinate frame W can either be defined by a force or a position goal. We estimate and inspect an admittance value. We use admittance [25] instead of impedance control to regulate the interaction forces, as many industrial manipulators do not support torque control. Simplified, an admittance gain can be estimated with the modelled velocity μ_v and force μ_F of a MP in TP-HSMM by

$$\dot{x} = K_{\mathrm{adm}}(F_{\mathrm{is}} - F_{\mathrm{set}})\ ,\ \hat{K}_{\mathrm{adm}} = \frac{\mu_\mathrm{v}}{\mu_\mathrm{F}}\ . \tag{10}$$

Position based movements are usually contact-free. They are characterized by high velocity v and small forces F, thus

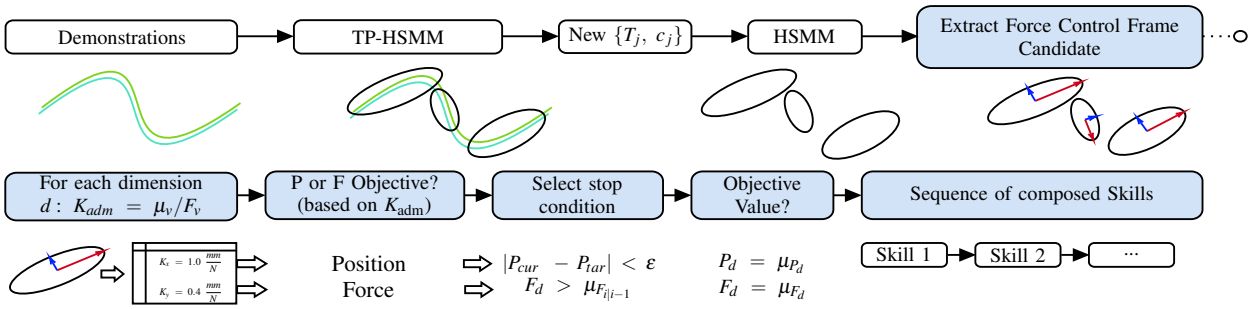

Fig. 3: Pipeline to convert demonstrations to *skills*. Novel steps of our approach are marked with shaded blue. New $\{T_j, c_j\}$ denotes here the *task parameters* with transformation T_j and offset c_j for each coordinate frame j given to the TP-HSMM.

$\hat{K}_{\text{adm}}$ will become large. Such movements denote a position goal and small values denote a force goal. However, in the case of very slow movements with no contact, sensor noise in force μ_{F} will exceed noise in μ_{v}, which results in low $\hat{K}_{\text{adm}}$, even though the goal should be a position goal. To address those cases, we introduce a hyperparameter to filter based on $\hat{K}_{\text{adm}}$. For a force goal, we only allow a gain between $0.1\,\frac{\text{mm}}{\text{N}}$ and $9\,\frac{\text{mm}}{\text{N}}$. If the admittance lies outside, the axis is considered a position goal. These values are chosen based on the materials used in the task and the Force-Torque-Sensor (FTS): We choose an upper bound of $9\,\frac{\text{mm}}{\text{N}}$ in order to reduce movements due to measurement noise of the FTS (0.1 N) to < 1 mm and a lower bound of $0.1\,\frac{\text{mm}}{\text{N}}$ as a force displacement of >10 N should result in a movement of > 1 mm.

C. Terminal Conditions $\mathcal{T}$

Several terminal conditions can be active at the same time. Here, we briefly describe those conditions.

Duration: An HSMM encodes the duration of each MP as a lognormal distribution with the parameters $(\mu_i^{\text{dur}},\ \sigma_i^{\text{dur}})$. We use the upper σ quantile of the lognormal to get an upper bound of the execution duration instead of using the mean directly. This increases robustness against disturbances.

Force: To extract a force terminal condition of a skill, we calculate the activation of two subsequent MPs, MP_i and MP_{i+1}. We then take the point, where both activations are equal. We combine the Gaussians of two MPs at their respective activation level (see Eq. 5). This results in a new Gaussian, from which we can get the mean as a *force threshold*. This condition is active only if $|\mu_{\text{F}}^{(i)}| < |\mu_{\text{F}}^{(i+1)}|$, i.e., if the force is increasing.

Position: For the Positioning stop condition of *skill* $\mathcal{S}_i$, we use the mean value μ_p of the encoded MP.

D. Execution: iTaSC Algorithm and Nullspace Projection

We determined a spacial description of force goals based on a new coordinate system, which we separated into force and position dominated axes. For execution, we use *pitasc*, an approach based on the task specification formalism *iTaSC* [10], [26]. The algorithm enables the definition of constraints (goals) in different coordinate systems and allows to calculate robot joint velocities. Conflicting constraints are resolved by the *task priority strategy* by Nakamura et al. [27], where subsequent tasks are projected into the nullspace of the higher prioritized tasks, in according to a hierarchy level

Fig. 4: Tested clamps. The clamps B and D feature a metal foot (higher assembly force), E, C and A a plastic foot.

$\mathcal{H}$. As the calculated goals are defined along each axis of an orthogonal coordinate system, there is no immediate conflict of goals. However, when a user defines additional behavior, conflicts will occur. In this case, the strategy allows to satisfy attribute 5 and 6, i.e., user defined behavior takes precedence and can be added easily. As this approach projects the position goals μ_{x} into the new coordinate frame W_{S_i} during execution, this step becomes redundant in preprocessing. There are no potential conflicting goals on orientation axes, as we don't consider torques, which is not needed in our use-case left for future work. The placement of orientation goals is thus arbitrary and we choose orientation as the least prioritized goal. It is worth mentioning that the approach is fully compliant with the *skill* structure suggested by Naegele et al. [11] and can be given a more descriptive name, e.g., *lin*, *push*, *pivot*, etc. or visualized, e.g., by Mixed-Reality (see [23]), which further improves interpretability.

V. Experiments and Results

We briefly describe the hardware components of the experimental setup. For further reference, please refer to [28].

We use a Franka Emika Panda, which is a 7-axis robot with torque control for each axis. To be compliant to industrial manipulators, we control its movements by sending joint velocity commands and do not make use of the joint-torque measurements. We track the location of the workpiece and the mounting rig (for the 3D experiments) with a common 2D camera (Logitech C920 Pro HD), to provide an unobtrusive way of recording data. To localize the 3D poses of the workpieces, we use the AprilTag v3 algorithm (see [28]). The poses of the workpiece and the mounting rig are determined relative to a reference AprilTag located near the robot (see Fig. 1). Underneath the mounting rig, a FTS of the type ATI AXIA 80 is placed. It measures forces at a rate of 1 kHz with a resolution of 0.1 N. In [28] we introduced our first use-case: assembly of terminal clamps onto a DIN rail. The main challenge in this task is to fix the terminal clamp to

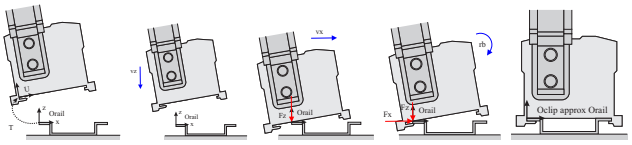

Fig. 5: Typical steps during the assembly of a terminal. (a) Preposition, (b) Align, (c) Contact and slide, (d) Pivoting and (e) Retreat

TABLE II: Resulting parameters for each *skill* $\mathcal{S}$. The name of the *pitasc skill* was added manually (see [11])

# S	name	F_x/N	F_y/N	F_z/N	$\hat{K}_x/\frac{mm}{N}$	$\hat{K}_y/\frac{mm}{N}$	$\hat{K}_z/\frac{mm}{N}$
0	*lin*	-	-	-	-	-	-
1	*lin*	-	-	-	-	-	-
4	*lin*	-	-	-	-	-	-
5	*insert*	-	-	-0.6	-	-	3.60
6	*force*	-4.10	0.11	1.91	0.41	0.30	0.20
7	*pivot*	-10.95	1.02	0.42	0.30	0.10	3.50
9	*push*	0.05	0.10	0.02	0.80	4.81	3.15

the DIN rail, which requires complex motions, such as force application and rotating of the clamp at the same time.

The typical process to assemble such a terminal clamp consists of these tasks (see Fig. 5):

(a) Picking up the terminal clamp,
(b) Aligning the terminal slightly tilted over the DIN rail,
(c) Establishing contact with the rail and sliding towards the rail while maintaining contact,
(d) Pivoting the terminal until it snaps onto the rail when the nose hits the end of the rail and
(e) Retreating the robot.

There is a great variety of clamps, here we consider the 5 types depicted in Fig. 4. We trained a TP-HSMM based on recorded data of the observation of 60 assemblies for Clamp A, which we demonstrated by hand and observed by camera and external FTS (see [28]), i.e., we do *not* employ kinesthetic teaching.

A. Adaptability to Variants

Due to the different geometries and materials of the clamps (see Fig. 4), each clamp needs its own model based on a separate set of demonstrations in the original implementation. For the first use-case, a TP-HSMM is trained only for clamp A. This model is adapted to a new position of the DIN rail and start position (see Sec. IV-A). The proposed segmentation approach results in a sequence of *skills* $\mathcal{S}_i$ seen in Tab. II. For $\mathcal{S}_5$, the admittance gain for the x and y coordinates fall outside the boundaries (see IV-B.1), resulting in a position controlled movement for those axes. *Skills* not listed in the table never reached a sufficient activation level. In addition, each *skill* is also defined by a position P and a orientation O (Fig. 2). The terminal conditions are *Position* ($\mathcal{S}_{0..4}$), Force ($\mathcal{S}_5$) and Duration ($\mathcal{S}_{5..9}$).

1) Adapting one robot program to different product variants: To evaluate the simplicity of adaptation for new variants, we apply the *skill*-model trained on clamp A to the clamps B, C, D and E. Initially the program fails for all of the clamps except for clamp A. We then adapt the program by changing the *skill* parameters. To adapt to the metal foot of clamps B and D, we first adapt the insertion force F_z for $\mathcal{S}_7$ (which describes the insertion rotation) to 5 N. Secondly, due to the different geometry of the variants, the start position of force-based skill $\mathcal{S}_6$ is misaligned. Therefor, we also adapt the x positions of $\mathcal{S}_4$ (pre-positioning) and $\mathcal{S}_5$ (establishing contact), which corresponds to the start position of $\mathcal{S}_6$. The results can be seen in Tab. III. Previously, additional demonstrations needed to be recorded for every variant (see [28]). For Clamp E, no demonstration data was available, but with the adaption it could still be assembled. The total adaption time for the program by an expert was $\sim$20 min. Recording 60 demonstrations for a new clamp requires on average (15.3 ± 0.3) min (based on $n = 6$ teach-ins) (plus training time for the model). Applying the proposed algorithm reduced teach time from $(15.3\,\text{min} \cdot (n-1)) = 76.5\,\text{min}$ to $(15.3\,\text{min} + 20\,\text{min}) = 35.3\,\text{min}$, resulting in a time saving of 41.2 min or 53.8 % of expert time. We showed that the assembly of all clamps (see Fig. 4) is enabled by a program derived from a LfD program only based on demonstrations of single clamp variant A. With our previous approach and other approaches, it is required to record new data for each variant. To adapt to new variants, only 3 basic and easy to understand parameters (1 x F / 2 x P) needed to be changed

TABLE III: Success-rates of the assembly program based on demonstration data from clamp A for the original version, skill-based without expert adoption, the adapted skill-based approach and adapted skill-based with error adaption to 2 cm.

Clamp	A	B	C	D	E
Pos. TP-HSMM [16], [28]	0/20	0/20	0/20	0/20	-
Force TP-HSMM [28]	20/20	0/5	0/5	0/5	0/5
skill-based	5/5	0/5	0/5	0/5	0/5
adapted skill-based	20/20	20/20	20/20	20/20	20/20
error adaption +2 cm	-	-	10/10	10/10	-

2) Increase Robustness by Expert Knowledge: A common error-case is the change of the goal position of the assembly. We evaluate this by applying a 2 cm offset in z direction to the rail. The original program results in a trajectory that moves close to the rail. Thus, if the rail position changes slightly, contact is established too early (if z position increases) or (otherwise) not at all. Although this can be handled with *task-parameterization*, it requires the recognition of the goal position (i.e., by a camera) and re-evaluating the HSMM at execution. This might not always be feasible, e.g., if the required position is not easily obtainable. We can increase the robustness of the program, by applying domain-specific knowledge, i.e., by increasing the z-position of $\mathcal{S}_{0,\ldots,5}$, the position-based *skills*, the program is able to assemble goal positions 2 cm higher than the original one (see the last row in Tab. III). Here, 4 parameters needed to be changed to achieve the adoption.

B. Transferability to new requirements

When requirements or environments change, robot programs often need to be adapted beyond what was anticipated before and what an automatic adaption can handle. In that case, a LfD program usually needs to be re-taught. However, industrial use-cases require experts to quickly do adaptions

TABLE IV: The results of the execution of an assembly program based on demonstration data from clamp *A*, with the tight stacking adaption.

Clamp Type with adaption	C	D
Gap	(0.59 ± 0.34) mm	(0.64 ± 0.20) mm
Success Rate	10/10	10/10

by themselves. Here, we test how our approach enables a user to adapt a LfD program when their training dataset is missing key data for a successful completion of a process.

Tight Stacking of Terminal Clamps: One requirement for stacking of clamps in a real terminal cabinet is space efficiency. This means that clamps have to be stacked very tightly. This behaviour was not demonstrated originally and is *not* reflected in the dataset. Instead of requiring new demonstrations, we show that it is easy to reuse the existing data and extend the application. We adapt the robot program from Tab. II by inserting a predefined *skill* from [11] after the execution of $\mathcal{S}_4$: $\mathcal{S}_{new}$, named *guarded approach*, which moves towards an already assembled clamp with a controlled velocity and stops when sensing a force. It stops at a force of $> 10\,\mathrm{N}$ and moves with $0.01\,\frac{\mathrm{m}}{\mathrm{s}}$. This *skill* establishes contact with the already assembled clamp. We then add a force goal in y-direction to reach 5 N for all subsequent $\mathcal{S}$ to keep the clamp into contact with the previous clamp (see schematics Fig. 6). We then test the program for a situation where one clamp of type E is already assembled and placed on the rail with distance of $\{8, 10, 12, ..., 26\}$ mm to its edge. We then use the program to assemble clamps C and D 10 times, respectively (results in Tab. IV). With the adaption, we can achieve a low gap of $\sim$0.6 mm, while still achieving 100% assembly rate. Overall, we only added *one* predefined *pitasc skill* with clearly defined behavior and one additional goal.

We show that it is possible to infuse an LfD program with expert knowledge, even if information about some dimensions is not available. For this, we use a 2D Dataset [29] for drawing letters, which was acquired in a completely different environment and contains no information about forces, end-effector orientation or the z-axis. This missing information is introduced at the *skill*-level by an expert without re-demonstrating. The original, HSMM-based program would have required either to re-teach the program with including the missing information or to augment the data with artificial values. The 2D data is processed with the proposed approach for 2D case and without forces. As the z axis is not defined, we add a goal to the z-axis to keep contact with the ground at 5 N, which is implemented similar to the approach in V-B. To establish contact, we add a *guarded approachskill* at the start. In addition, the letters could be easily scaled by expert knowledge, by adding an scaling to the x and y goals or adapted for different pens by altering the force. By adding two default *pitasc skills* with 3 parameters (2 x F, 1 x vel), a demonstrated 2D trajectory *without* force control can be expanded to a 3D trajectory with force-control.

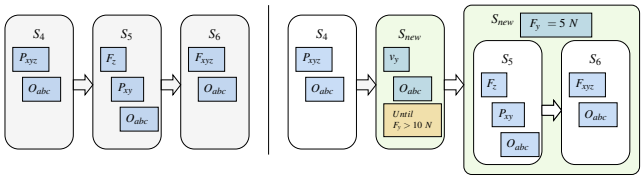

Fig. 6: Adoption done by hand for *tight-stacking* use-case. Left: The original program after it was learned (partial). Right: Adapted program. new elements in green.

VI. Conclusion and Outlook

In this work we evaluated an approach to automatically transfer an LfD model based on force, position and orientation trajectories to a *skill*-based model. We show that *skill* constraints, especially of force-based *skills*, can be automatically derived from a probabilistic LfD model, providing an interface to the operator on level 3, making it easier to understand and versatile enough to be adapted. We improve the reusability of programs by enabling human operators to include additional contraints to different variants (Sec. V-A.1), partially different behaviour (Sec. V-B) and make them more robust (Sec. V-A.2). We achieve this goal by improving the readability of the program by segmenting it into several, easier-to-understand *skills*. Furthermore, we show that demonstration data from different domains such as 2D Data can be adapted by including expert knowledge to fill in the gaps (Sec. V-B). With this, it is possible to use force control without recording force data directly. This manual adaption comes at the cost of losing the abstraction of the *task-parametrization* of the HSMM model. This might result in failure in some cases that can't be easily handled manually, e.g., obstacles might not be avoided anymore after an environment change. We assume this method to be general, given that additional information can be adapted by the user quickly. We showed that we can handle inaccuracies of FTS and vision by fusing them in our previous work [28]. In this work, we showed that varying stiffness of different variants can be handled by quickly including expert knowledge. One downside of this approach is, that an expert is required. This implies, that its value and time-improvement is based upon the knowledge of the expert as well as their familiarity with the given framework. An experimental evaluation of the framework's and presentation's usability is subject of future work.

In future work, robot *skills* should be further abstracted. The representation in this paper can be matched to a predefined library of *skills* provided by pitasc, which could improve readability even further. The representation of *skills* can be combined with Mixed-Reality (see [23]), which provides a powerful tool to manipulate the learned robot *skills*.

References

[1] International Federation of Robotics, "Executive summary world robotics 2021 industrial robots," *World Robotics Report*, pp. 12–16, 2021.

[2] L. Halt, F. Naegele, P. Tenbrock, and A. Pott, "Intuitive constraint-based robot programming for robotic assembly tasks," in *2018 IEEE Int. Conf. on Robotics and Automation (ICRA)*, May 2018. doi: 10.1109/icra.2018.8462882

[3] F. Nagele, L. Halt, P. Tenbrock, and A. Pott, "A prototype-based skill model for specifying robotic assembly tasks," in *2018 IEEE Int. Conf. on Robotics and Automation (ICRA)*, May 2018. doi: 10.1109/icra.2018.8462885

[4] F. Steinmetz, V. Nitsch, and F. Stulp, "Intuitive task-level programming by demonstration through semantic skill recognition," *IEEE Robot. Autom. Lett.*, vol. 4, no. 4, pp. 3742–3749, Oct. 2019.

[5] J. Albus, R. Lumia, J. Fiala, and A. Wavering, "Nasrem – the nasa/nbs standard reference model for telerobot control system architecture." Proceedings of the 20th International Symposium on Industrial Robots, Tokyo, 1, JA, 1989-10-06 00:10:00 1989.

[6] T. Hasegawa, T. Suehiro, and K. Takase, "A model-based manipulation system with skill-based execution," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, pp. 535–544, 1992. doi: 10.1109/70.163779

[7] M. T. Mason, "Compliance and force control for computer controlled manipulators," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 11, no. 6, pp. 418–432, 1981. doi: 10.1109/tsmc.1981.4308708

[8] I. Weidauer, D. Kubus, and F. M. Wahl, "A hierarchical extension of manipulation primitives and its integration into a robot control architecture," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, may 2014. doi: 10.1109/icra.2014.6907653

[9] C. Samson, M. Le Borgne, and B. Espiau, "Robot control (the task function approach)," *Robotica*, vol. 9, no. 4, pp. 447–448, dec 1991. doi: 10.1017/s0263574700000643

[10] J. D. Schutter, T. D. Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx, "Constraint-based Task Specification and Estimation for Sensor-Based Robot Systems in the Presence of Geometric Uncertainty," *The Int. Journal of Robotics Research*, vol. 26, no. 5, pp. 433–455, 2007. doi: 10.1177/0278364907078091O7

[11] F. Naegele, L. Halt, P. Tenbrock, and A. Pott, "Composition and incremental refinement of skill models for robotic assembly tasks," in *2019 Third IEEE Int. Conf. on Robotic Computing (IRC)*, 2019. doi: 10.1109/IRC.2019.00034 pp. 177–182.

[12] O. Kroemer, S. Niekum, and G. Konidaris, "A review of robot learning for manipulation: Challenges, representations, and algorithms," *Journal of Machine Learning Research*, vol. 22, no. 30, pp. 1–82, 2021.

[13] P. Pastor, M. Kalakrishnan, L. Righetti, and S. Schaal, "Towards associative skill memories," in *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, nov 2012. doi: 10.1109/humanoids.2012.6651537

[14] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artificial Intelligence Review*, vol. 11, no. 1/5, pp. 11–73, 1997. doi: 10.1023/a:1006559212014

[15] S. Calinon, "A tutorial on task-parameterized movement learning and retrieval," *Intelligent Service Robotics*, vol. 9, no. 1, pp. 1–29, Sep 2015. doi: 10.1007/s11370-015-0187-9

[16] S. Calinon and D. Lee, "Learning control," in *Humanoid Robotics: a Reference*, P. Vadakkepat and A. Goswami, Eds. Springer, 2019, pp. 1261–1312.

[17] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, "Probabilistic movement primitives," in *Advances in Neural Information Processing Systems*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., vol. 26. Curran Associates, Inc., 2013.

[18] C. A. V. Perico, J. D. Schutter, and E. Aertbelien, "Combining imitation learning with constraint-based task specification and control," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1892–1899, Apr 2019. doi: 10.1109/lra.2019.2898035

[19] J. Silverio, S. Calinon, L. Rozo, and D. G. Caldwell, "Learning task priorities from demonstrations," *IEEE Transactions on Robotics*, vol. 35, no. 1, pp. 78–94, feb 2019. doi: 10.1109/tro.2018.2878355

[20] A. Laemmle, T. Koenig, M. El-Shamouty, and M. F. Huber, "Skill-based programming of force-controlled assembly tasks using deep reinforcement learning," *Procedia CIRP*, vol. 93, pp. 1061–1066, 2020. doi: https://doi.org/10.1016/j.procir.2020.04.153 53rd CIRP Conf. on Manufacturing Systems 2020.

[21] L. Johannsmeier, M. Gerchow, and S. Haddadin, "A framework for robot manipulation: Skill formalism, meta learning and adaptive control," *2019 International Conference on Robotics and Automation (ICRA)*, 2019. doi: 10.1109/icra.2019.8793542

[22] M. Sharma, J. Liang, J. Zhao, A. Lagrassa, and O. Kroemer, "Learning to compose hierarchical object-centric controllers for robotic manipulation," in *Proceedings of the 2020 Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, J. Kober, F. Ramos, and C. Tomlin, Eds., vol. 155. PMLR, 16–18 Nov 2021, pp. 822–844.

[23] J. Krieglstein, G. Held, B. A. Bálint, F. Nägele, and W. Kraus, "Skill-based robot programming in mixed reality with ad-hoc validation using a force-enabled digital twin," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023. doi: 10.1109/ICRA48891.2023.10161095 pp. 11 612–11 618.

[24] D. Bruno, S. Calinon, and D. G. Caldwell, "Learning autonomous behaviours for the body of a flexiboctle surgical robot," *Autonomous Robots*, vol. 41, no. 2, pp. 333–347, Jan 2016. doi: 10.1007/s10514-016-9544-6

[25] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer London, 2009.

[26] R. Smits, T. D. Laet, K. Claes, H. Bruyninckx, and J. D. Schutter, "iTASC: a tool for multi-sensor integration in robot manipulation," in *2008 IEEE Int. Conf. on Multisensor Fusion and Integration for Intelligent Systems*, Piscataway, NJ, 2008. doi: 10.1109/MFI.2008.4648032. ISBN 978-1-4244-2143-5 pp. 426–433.

[27] Y. Nakamura, H. Hanafusa, and T. Yoshikawa, "Task-priority based redundancy control of robot manipulators," *The International Journal of Robotics Research*, vol. 6, no. 2, pp. 3–15, jun 1987. doi: 10.1177/027836498700600201

[28] D. Bargmann, P. Tenbrock, L. Halt, F. Naegele, W. Kraus, and M. F. Huber, "Unobstructed programming-by-demonstration for force-based assembly utilizing external force-torque sensors," in *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2021. doi: 10.1109/smc52423.2021.9658707

[29] S. Calinon and E. Pignat. PbDlib library. [Online]. Available: https://gitlab.idiap.ch/rli/pbdlib-python/