Intuitive Task-Level Programming by Demonstration through Semantic Skill Recognition

Steinmetz, Franz; Nitsch, Verena; Stulp, Freek

Keywords: Human-Centered Automation, Learning from Demonstration, Intelligent and Flexible Manufacturing, Cognitive Human-Robot Interaction

Intuitive Task-Level Programming by Demonstration through Semantic Skill Recognition

Franz Steinmetz[1], Verena Nitsch[2], and Freek Stulp[1]

***Abstract*—Intuitive robot programming for non-experts will be essential to increasing automation in small and medium-sized enterprises (SMEs). Programming by Demonstration (PbD) is a fast and intuitive approach, whereas programs created with Task-Level Programming (TLP) are easy to understand and flexible in their execution. In this paper, we propose an approach which combines these complementary advantages of PbD and TLP. Users define complete task-level programs including all parameters through PbD alone. Therefore, we call this approach Task-Level Programming by Demonstration (TLPbD). TLPbD extends skill-based approaches by enabling experts to semantically annotate robot skills with their conditions and effects, which facilitates online skill recognition from pure demonstrations by a non-expert. In a user study with 21 participants, the approach is compared with an existing intuitive TLP approach. The results show that the new approach drastically reduces the programming time while at the same time being more intuitive, reducing mental load, and achieving the same or even better skill sequences.**

***Index Terms*—Human-Centered Automation, Learning from Demonstration, Intelligent and Flexible Manufacturing, Cognitive Human-Robot Interaction**

I. Introduction

INTUITIVE robot programming for non-experts will be essential to increasing automation and productivity in *small and medium-sized enterprises* (SMEs). For instance, kinesthetic teaching of *collaborative robots* (cobots) to perform repetitive and physically demanding tasks would alleviate shop-floor workers. A growing number of companies develop therefore software that enables robot task programming without requiring expertise in programming or robotics. For instance, *graphical user interfaces* (GUIs) use *wizards* to guide the user in composing, sequencing and parameterizing existing robot skills to solve a task. Examples for such *Task-Level Programming* (TLP) approaches are Franka Desk [1], ArtiMinds RPS [2], and our own software RAZER [3].

TLP is intuitive, as skills and the subgoals they achieve correspond to a user's mental model of how tasks are solved. The resulting programs are also robust (as error-handling can be built into the individual skills), safe (as individual skills can be certified) and general (as individual skills can be parameterized for variations of a task). A disadvantage is that TLP, though intuitive, can be time-consuming.

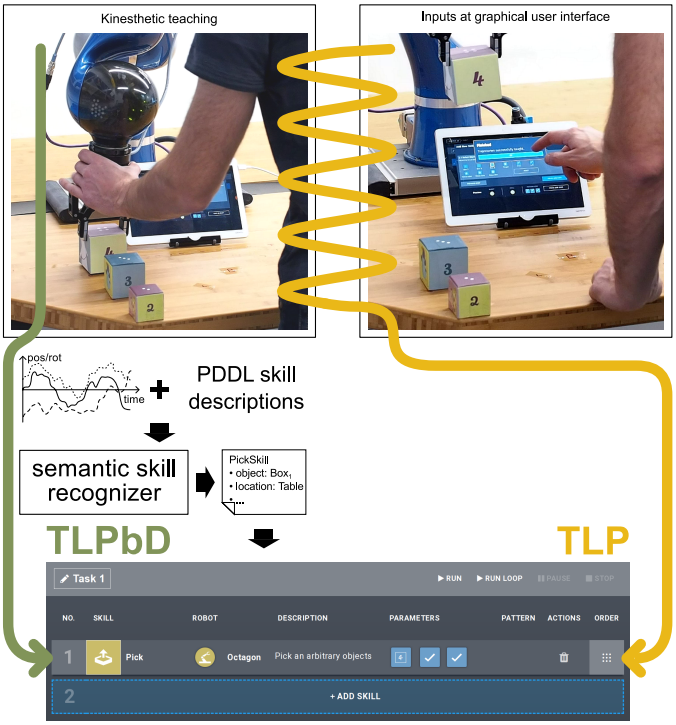

Fig. 1. When using TLP (orange line, right side), the user often has to switch between inputs at a GUI and demonstrations at the robot. With the presented TLPbD approach (green line, left side), pure kinesthetic teaching is sufficient for specifying task-level programs. Thy system automatically recognizes online the demonstrated skills and parameters using PDDL descriptions of available skills.

An alternative to non-expert programming of robots is *Programming by Demonstration* (PbD), e. g. by guiding the robot through kinesthetic teaching [4]. This approach is intuitive and fast. A disadvantage is that the recorded trajectories do not represent semantic knowledge about the underlying task, e. g. its ultimate aim, the relevant subgoals and subtasks, etc. This makes it more difficult to generalize the motion, and also causes a mismatch between the user's representation (aims and subgoals) and that of the robot (trajectories).

The main contribution of this paper is to present a programming concept for cobots that merges the complementary advantages of TLP and PbD, which we call *Task-Level Programming by Demonstration* (TLPbD), as illustrated in Fig. 1. In TLPbD, the only input modality is PbD. But the output

Manuscript received: February, 21, 2019; Revised May, 29, 2019; Accepted June, 27, 2019.

This paper was recommended for publication by Editor Allison M. Okamura upon evaluation of the Associate Editor and Reviewers' comments. This work was funded by the European Commission through the project H2020 AnDy (GA no. 731540), and the "Factory of the Future" project of the German Aerospace Center (DLR).

[1] German Aerospace Center (DLR), Robotics and Mechatronics Center (RMC), Münchner Str. 20, 82234 Weßling, Germany. Contact: franz.steinmetz@dlr.de

[2] RWTH Aachen University, Institute of Industrial Engineering and Ergonomics (IAW), Bergdriesch 27, 52056 Aachen, Germany. Contact: v.nitsch@iaw.rwth-aachen.de

Digital Object Identifier (DOI): 10.1109/LRA.2019.2928782.

result (and also the visual feedback during programming) is a task-level program including all parameters. We achieve this by the automatic recognition of skills and their parameters online during the demonstration. With TLPbD, programming is fast and intuitive (advantages of PbD), and the generated program is easy to understand and flexible in the execution (advantages of TLP). This makes the approach highly suitable for industrial scenarios.

As an example, let us describe the process of appending a Pick skill to an existing task: First, the user moves the gripper of the robot to a desired object (e. g. Box_1), closes the gripper (using a foot pedal) and lifts the box. Thanks to *Planning Domain Definition Language (*PDDL) descriptions of the available skills, the system correctly identifies the demonstration online as a pick operation and adds it with the object parameter set to Box_1 to the current task.

Further contributions are the implementation of the approach, its integration into our software architecture RAZER, and application to a typical low-batch-size scenario. Finally, we empirically evaluate the advantages of our TLPbD approach over pure TLP in a user study with 21 subjects.

II. Related Work

Programming by Demonstration is a popular approach to teach a robot skills [4]. Methods such as *teleoperation*, *observational learning* and *kinesthetic teaching* allow for natural and intuitive demonstrations of trajectories and force profiles. Popular approaches for modeling the recorded motions into *motion primitives* include *Dynamic Movement Primitives (*DMPs) [5] and *Gaussian Mixture Models (*GMMs) [6].

Various techniques have been developed that extend the basic replay of trajectories recorded during PbD. For instance, it is possible to adapt a learned motion velocity or final position [7]. Furthermore, important task frames can be identified, which allows to execute the motion relative to varying coordinate systems [6]. With several trajectories for the same demonstration, [8] showed how motions can be automatically segmented and collected in a movement primitive library. Meier et al. [9] probabilistically recognize motion primitives online during the recording. One shortcoming of motion primitive approaches is that the primitives, being based on trajectories, lack an inherent logic and semantics resulting from a deeper understanding of the underlying problem. Furthermore, they usually do not include failure-handling and safety verification routines.

Verifying safety and reliability is more straight-forward with skill-based approaches, where a robot expert predefines and generically implements the capabilities of a robot system into specific software modules called *skills* [10]–[13]. Skills are executable programs with logic for e. g. strategies and failure-handling designed for solving specific tasks (such as peg-in-hole), and are crafted by an expert. This logic may be represented in various forms, such as Behavior Trees [14], state machines [3] and specialized controllers [11].

Task-Level Programming builds on the skills developed for a robot, by providing user interfaces (e. g. a GUI) to non-experts to create tasks consisting of skills [1]–[3], [11], [14], [15]. These interfaces enable users to sequence the appropriate skills and specify relevant parameter values in order to tailor them to the task at hand. As a visual programming language, task-level approaches are superior to textual programming languages regarding its intuitive use [16].

The user interfaces of TLP approaches have come in various forms. Two commercial software products, already used in industries are Franka Desk [1] and ArtiMinds RPS [2]. Two examples for research software prototypes are described in [11], [12]. Both allow skills to be composed through drag & drop, and parameterized through kinesthetic teaching. The work described in this paper builds on RAZER [3], our framework for TLP. Another example is CoSTAR, which utilizes Behavior Trees [14]. CoSTAR lowers the complexity for the user by increasingly integrating the logic into the available building blocks. The programming paradigm in [15] combines TLP with semantic process descriptions. By exploiting domain knowledge and CAD models, tasks can be intuitively specified.

Note that PbD can be used as a specific interaction modality *within* TLP. That is, most interactions are through a GUI, but occasionally skills will be parameterized by physical interaction with the robot in gravity compensation mode, e. g. to demonstrate the location where drilling should take place [3], [11]–[13].

Even with the most refined interfaces, creating a robot program using PbD within TLP can still be cumbersome. It requires many clicks with the mouse (or tabs on a touch device), frequent switches between the programming device and the robot, and context switches between physical interaction with the robot and focused reading and following of instructions on the screen (see Fig. 1).

Four further input modalities besides PbD (touch, gesture, speech and 3D tracking device) have been evaluated in [17] for their perceived suitability for different parameter types. Which modality is preferred depends both on the user and the type of the parameter. We leave the choice of the input modality (PbD or touch) up to the user.

PbD and TLP are both powerful approaches, with complementary advantages. PbD is more intuitive and efficient to use, as there are no context switches for the user. But programs created with TLP are easier to read and modify and more appropriate to be executed in an industrial context. Therefore, with TLPbD, we propose a combination where PbD is the *only* input modality, but the movements a user makes are automatically translated into parameterized skills. The result of the interaction is a sequence of parameterized skills, presented in a GUI for further editing and execution in industrial tasks. The user essentially performs Task-Level Programming *through* Programming by Demonstration.

Using PbD to infer previously known and described actions has previously been proposed in [18]. Hereby, *Object-Action Complexes (*OACs) and *semantic event chains (*SECs) are utilized. However, the approach is limited in the definition of conditions and effects, as only object relations (touching or non-touching) can be specified. In contrast to our approach, this approach works offline. PbD in combination with a GUI and voice commands is used in [19], enabling users to define new skills which can later be used in tasks. They use semantics

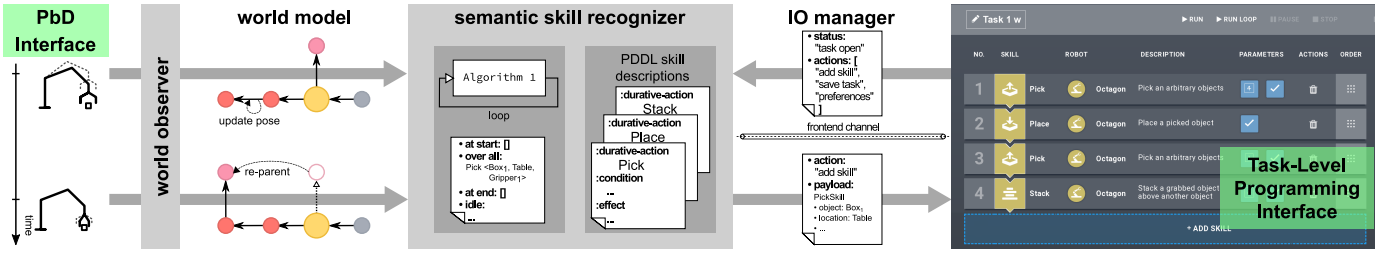

Fig. 2. The diagram depicts the main components of the semantic skill recognition system and the communication between them. The semantic skill recognizer sends parameterized skills to the TLP interface, when the latter signalizes to be in the correct status. The recognizer creates the skills based on the world model, updated by the world observer, and the PDDL descriptions of the available skills.

in the sense that all primitives and parameters are typed.

As an alternative to enabling users specify the process of how to reach a goal, automated task planning tools create a plan from a goal by respecting given conditions. This has been combined with PbD to propose possible solutions to the planner [20] and to identify constraints [21].

III. Semantic Skill Recognition

An overview of our Task-Level Programming by Demonstration approach is shown in Fig. 2. The user interacts with the robot through kinesthetic teaching as a PbD interface. The core component in our approach is the *semantic skill recognizer*, which extracts skills and their parameterization online during the interaction. To do so, the algorithm uses not only the raw trajectories, but also more abstract representations stored in a *world model*. During the interaction, the world model is continually updated by the *world observer*. The semantic skill recognition algorithm queries the world model for relationships between objects (connections, relative and absolute poses). These relationships are used to recognize and select skills based on their conditions and effects, which are annotated in PDDL [22]. Finally, the TLP interface shows the sequence of skills which have been recognized so far. Thus, the input modality of our approach is PbD, but the result and visual feedback is given with a TLP interface. In the following sections, we describe the implementation of the components in Fig. 2 in more detail.

A. Robot PbD Interface

The main mode of interaction with the LWR IV+ robot used in the evaluation is kinesthetic teaching of tasks in gravity compensation mode. Furthermore, a foot pedal with buttons was used as an interface to the robot. The left button toggled the gravity compensation mode between on and off. The right button causes the gripper to open and close. For example, the user presses the left pedal to bring the robot in gravity compensation mode, moves the gripper above Box_1, uses the right pedal to close the gripper (grasping the box) and lifts the box with the robot arm.

B. World model

The semantic skill recognition algorithm makes frequent queries about conditions that hold in the current state of the world. To facilitate and speed up these queries, we maintain a world model as an abstraction layer to the continuous data streams in the real world.

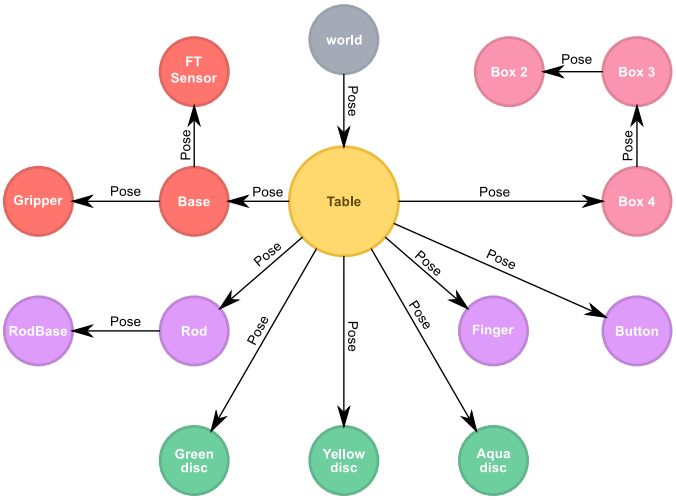

Fig. 3. The world model used for the user study. The colors indicate the type. Yellow stands for SURFACE, red for ROBOT, green for HOLLOWCYLINDER, purple for CYLINDER, and rose for CUBOID. In this instance of the world, Box 2 is stacked on Box 3, which itself is stacked on Box 4.

The world model is implemented using a Neo4j graph database. Its nodes represent entities in the world and the edges are the relative poses between them. The world model used in the user study (Section IV-A) is visualized in Fig. 3.

There are different types of nodes with distinct properties and methods. Node types can inherit from each other. All nodes (except for the root) inherit from the type RIGIDBODY, and by this automatically carry information e. g. about their bounding box. Methods of nodes can be used to further process and query this information.

If predicates of conditions or effects are evaluated, they are grounded on the data in the world model, not the real world data (except for readings from the force-torque sensor). Parameters of predicates are always nodes within the world. Several predicates can be found in Listing 1, the PDDL description of the Peg-in-Hole skill. Consider as an example the predicate `(close_to ?object ?gripper)`. The first parameter is `?object`, which is a node of type WORLDMODEL.HCR.OBJECT (see parameter definition, types in PDDL may not contain dots). When the predicate is evaluated, the method `close_to` of the instantiated Object node is called with the instantiated Gripper node as parameter: `object.close_to(gripper)`. The method `close_to`

calculates and checks the distance between the bounding boxes of the two nodes. This way, predicates are only limited by the implementation of the methods. One cannot only write predicates or methods based on geometric information, but also access sensor data, such as force-torque sensor readings.

A separate process, the *world observer*, is responsible for keeping the world model up to date, after it has been initialized at the beginning with the correct data. For this, the robot pose and the gripper status are checked frequently (>50 Hz). The robot pose is directly used in the world model as relative pose between the Base and the Gripper node (see Fig. 3). If the gripper was closed and signals that it is grasping something, the node of the object closest to the gripper is attached to the node of latter with the current relative pose stored in the edge. Upon opening the gripper, the node of the previously grasped object is attached to the node of the closest surface or object. If no surface or object is nearby, the node is detached.

C. Skills

The aim of our work is to extract a sequence of skills from the PbD interaction with the robot. For this, six skills have been provided for the evaluation (Section IV), namely Pick, Place, Stack, Peg-in-hole, Push and Hand-over. Before explaining the semantic skill recognizer, we first describe the underlying skill representation it builds upon. A general description of skills in the context of TLP can be found in Section II and [10]. Skills in our RAZER framework follow these principles, in that skills have an execution block and parameters [3]. These parameters have names and types. The execution logic of the skills is implemented with RAFCON[1] [23].

The only difference to the skill design as described in [3] is the additional semantic skill description in form of PDDL. PDDL 2.1 was enriched with so-called *durative actions* [24]. Durative actions can have *timed conditions*, which explicitly hold AT START, AT END or OVER ALL. These actions can also have *timed effects* that take place AT START or AT END. The PDDL annotations of skills symbolically describe their conditions and effects. Parameters specify the variables and their types used within conditions and effects. Each parameter in the semantic description maps to a parameter of the skill. As an example, see the PDDL description of the Peg-in-hole skill in Listing 1. To parse such PDDL strings, the open-source project pddl-lib[2] was extended to support durative actions.

D. Semantic skill recognizer algorithm

The skill recognizer monitors changes in the world (caused by PbD), and maps them to the conditions and effects of all available skills. This is necessary to determine which skill is active, and what its parameters are. To do so, the recognizer identifies all possible combinations of parameter values for each skill. Consider for example the Peg-in-hole skill in Listing 1 with the parameters `?object` and `?target`, both of type OBJECT, and `?gripper` of type LOCATION. If there

[1]https://github.com/DLR-RM/RAFCON/

[2]https://github.com/hfoffani/pddl-lib

```
(define (domain peg-in-hole-domain)
  (:durative-action peg-in-hole
    :parameters (
      ?object - worldmodel_hcr_Object
      ?target - worldmodel_hcr_Object
      ?gripper - worldmodel_hcr_Gripper)
    :condition (and
      (at start (not (equal ?object ?target )))
      (at start (not (at ?object ?target )))
      (at start (not (bounding_box_intersection ?object ?target )))
      (at start (at ?object ?gripper ))
      (at start (grasping ?gripper ))
      (at start (close_to ?object ?target ))
      (at start (close_to ?object ?gripper ))
      (over all (close_to ?object ?target ))
      (over all (close_to ?object ?gripper )))
    :effect (and
      (at end (at ?object ?target ))
      (at end (bounding_box_intersection ?object ?target ))
      (at end (not (at ?object ?gripper )))
      (at end (not (grasping ?gripper )))
      (at end (not (close_to ?object ?gripper ))))))
```

Listing 1. PDDL definition of the Peg-in-hole skill

exists ten objects and one gripper (as in our world model, see Fig. 3), then in sum $10 \times 10 \times 1 = 100$ combinations are possible. The skill could be instantiated with each of these combinations. We call a skill together with a certain combination of values a *skill-value combination (*SVC).

The skill recognizer checks for changes in the world that could lead to a status change of a SVC at approximately 30 Hz (see Section IV-C for details on the performance). The recognizer keeps a list with all SVCs and their status, which is either IDLE, AT START, OVER ALL, or AT END. Initially, all SVCs are IDLE. In each iteration, the status of all SVCs is updated according to Algorithm 1. If, e. g., a SVC is currently IDLE and the AT START conditions of the skill are fulfilled for the value combination, then its status is set to AT START (see Lines 15 to 17). If a SVC reaches the status AT END, it is assumed that the according skill has just been demonstrated with the corresponding values. The TLP interface is notified (see Section III-E for details) and the SVC is reset to IDLE (see Lines 5 to 9).

```
 1: procedure LOOP
 2:     for skill in skills do
 3:         for SVC in skillValueCombinations(skill) do
 4:             if SVC.status = OVER ALL and ↩
                       effectOccured(SVC, AT END) then
 5:                 SVC.status ← AT END
 6:                 skill_parameterized = parameterize(SVC)
 7:                 append(task, skill_parameterized)
 8:                 SVC.status ← IDLE
 9:                 break
10:             if SVC.status = AT START or ↩
                       SVC.status = OVER ALL then
11:                 if conditionFulfilled(SVC, OVER ALL)
12:                     SVC.status ← OVER ALL
13:                 else
14:                     SVC.status ← IDLE
15:             else if SVC.status = IDLE
16:                 if conditionFulfilled(SVC, AT START)
17:                     SVC.status ← AT START
```

Algorithm 1. Skill recognizer loop

The algorithm uses some additional functions. The function `skillValueCombinations` determines all SVCs for a

given skill. `conditionFulfilled` checks if the specified timed condition of the given skill currently holds. Likewise, `effectOccured` tests if the specified timed effect of the given skill took place.

The recognizer can evaluate arbitrary condition and effect statements. For example, the `not` operator leads to a negation and if multiple AT START conditions are defined, their boolean results are bitwise ANDed together. The predicates of conditions and effects are grounded using the world model, as it is described in Section III-B.

E. Communication with TLP interface

The semantic skill recognizer directly communicates with the TLP interface of RAZER, called frontend [3]. This is where the "TLP" in "TLPbD" comes from. When a task (a robot program) is open in the frontend, it notifies the skill recognizer that it currently accepts skills. The recognizer automatically starts the detection loop (Algorithm 1). If a skill has been identified, it is propagated to the frontend.

This communication is achieved using a component called the *input/output manager (*IO manager), which provides dedicated communication channels. All components can publish their current status (e. g. "task open") together with possible actions (e. g. "add skill", "save task") over a channel and receive action requests from other components on the same channel. For this, a component (in this case the frontend) opens a new channel on which it sends its status together with all currently available actions or commands. Other components (in this case the semantic skill recognizer) can listen on that channel and trigger an applicable action, such as "add skill", together with some payload, which would for this command be the parameterized skill.

The frontend expects a parameterized skill entity for the task. For this, the skill recognizer needs to convert the identified SVC. This is done by the `parameterize` function (Algorithm 1, Line 6). It creates a new skill and sets the parameters using the corresponding values from the SVC. In addition, skills can have parameters with special names "trajectory" and "force_profile". If these are present, they retrieve the robot data recorded during the time interval of the skill demonstration. A `ParameterDeducer` (see [3]) extracts from this additional information relevant for the execution, such as pre-pick or post-place poses. After the parameterization, the `append` function (Line 7) sends the skill as payload of the "add skill" action to the frontend using the IO manager as described above.

F. Integration in RAZER

Our implementation builds upon the RAZER architecture [3]. If a skill is added to the task by the semantic skill recognizer as described in Section III-E, it appears in the TLP interface (the frontend) as if it was added manually using the skill wizard of the GUI.

This has the advantage that the user profits from all features of the TLP. For instance, if after the demonstration the user decides that a slightly different order of the skills may be more efficient in practice, she can simply drag & drop the skill in the GUI. This is much more efficient than showing the updated sequence again through PbD. If a skill parameter, such as the object to pick, needs to be modified to generalize to a new task (or because the object was not recognized correctly during the demonstration), the skill can be opened and the parameter be adjusted with a single click. Therefore, TLPbD inherits all the advantages of TLP.

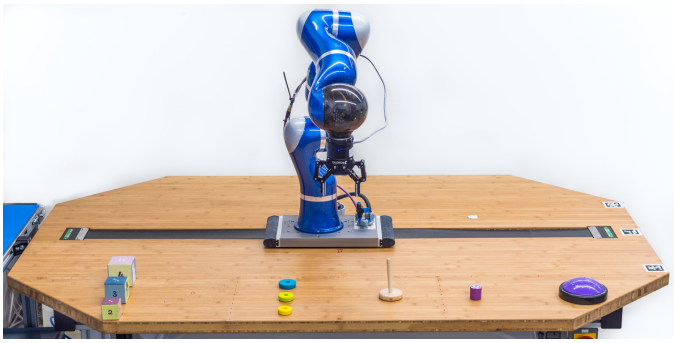

Fig. 4. The setup of the user study. The left area is for task 1 (box stacking), the middle for task 2 (peg-in-hole) and the right for task 3 (push button). The LWR is mounted on a linear axis for greater reach.

IV. Evaluation

To evaluate the system, we conducted an experimental user study and tested the scalability of the system. The setup for the study is shown in Fig. 4.

A. User Study

This user study compares the existing TLP approach [3] to the TLPbD approach proposed in this paper[3]. Our aim was to determine how the skill recognition system in the TLPbD approach supports the user in the programming process. Our hypothesis was that the programming time is reduced, while at the same time the usability is improved.

Using an a priori power analysis ($\alpha = 0.05$, large expected effect size), we determined the minimum sample size to be 20. The user study had 21 participants, 4 female and 17 male, aged between 25 and 59. All stated to have experience with touch devices. The expertise with robots and PbD was mixed and ranged between "no experience at all" and "highly experienced". The perceived technical affinity of each user regarding *enthusiasm* and *expertise* was measured using a subset of the TA-EG questionnaire [25].

The procedure of the study was as follows. A two-minute introduction was given to every participant for the usage of the tablet interface, the gravity compensation mode and the gripper. Then, the participants were asked to perform three different tasks, once with TLP and once with TLPbD (in random order, within-subject design). After performing the first task of each approach, the users filled in two questionnaires. The *questionnaire for the subjective consequences of intuitive use (*QUESI) was used to measure the intuitive use with the five subscales *subjective mental load*, *perceived achievement of*

[3]Note that our TLP approach (described in detail in [3]) makes use of PbD as an interaction modality for the specification of single parameters. But this is PbD *within* TLP, and not TLPbD, as discussed in Section II.

Fig. 5. The parameterized skill sequences of task 1 (top), task 2 (middle) and task 3 (bottom) in the RAZER TLP interface. A tick stands for a specified trajectory. The sequences are the same for both approaches.

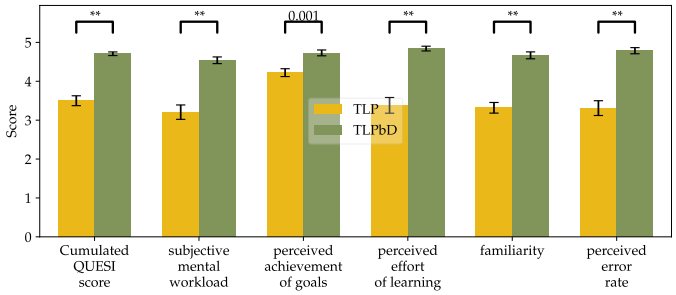

Fig. 6. Results regarding the intuitive use determined by the QUESI. The plot shows the mean of the overall score and the subscales, for TLP (orange) and TLPbD (green). The scores are values between 1 (worst) and 5 (best). The value above the brackets is the asymptotic significance p, ** means $p \ll 0.001$. ANCOVA was used for the calculations of the variance and significances with the technical enthusiasm and expertise as covariates.

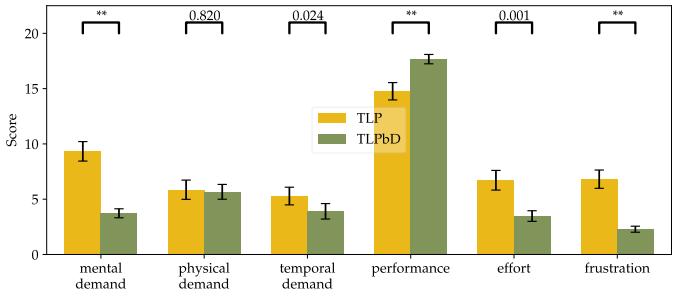

Fig. 7. Mean values regarding the workload-related factors using the NASA TLX. The TLX score is a value between 1 and 20. Except for performance, lower values are better. See Fig. 6 for details on the bracket values and the calculations.

goals, *perceived effort of learning*, *familiarity*, and *perceived error rate* [26], [27]. This was complemented by the NASA *Task Load Index (*TLX) measuring the workload-related factors *mental demand*, *physical demand*, *temporal demand*, *effort*, *performance* and *frustration* [28]. Furthermore, the times the users required for programming each task was measured. At the end, the participants could give further oral feedback. Fig. 5 shows the desired skill sequences for the following three used tasks:

1) Stack the three boxes on top of each other (from large to small)
2) Slide three rings onto the rod (from large to small)
3) Grasp the cylinder, push the button with it twice, and hand it over

The experimental setup of the tasks is pictured in Fig. 4. A demonstration of these tasks with a comparison between TLP and TLPbD can be found in the accompanying video.

B. Results

The results regarding the intuitive use of the system are summarized in Fig. 6. On average, users rated the intuitive use of TLPbD 35% better than TLP. People felt more familiar with the approach (by 41%) while requiring less mental workload (42%).

Fig. 7 summarizes the results regarding the workload-related factors of the TLX. While the physical demand for both approaches is similar, there is a statistically significant improvement on all other scales. For example, the frustration is reduced by 66% and the mental demand by 59%, which is in accordance to the results of the QUESI.

Regarding the results of the QUESI and TLX, we were interested whether the perceived technical affinity had any influence on the different scores. We used *analysis of covariance (*ANCOVA) to determine the partial eta-squared (η_p^2) values as measure for the effect size. For enthusiasm, we get an average effect size of 0.019 (TLP) and 0.035 (TLPbD); for expertise we get 0.172 (TLP) and 0.060 (TLPbD). Thus, the technical enthusiasm has less influence than the technical expertise and there is less influence of the expertise on the TLPbD approach than on the TLP approach.

The mean programming times for the two approaches and three tasks are plotted in Fig. 8. The times for task 2 and 3 were normalized to task 1, i. e. the different requirements of each task are respected by measuring the time for each task required by an expert with no learning effect. The results showed two statistically significant effects ($p \ll 0.001$). First, using TLPbD, the time is reduced by a factor of over 5. Second, there is a learning effect for both approaches. This effect mainly took place between task 1 and 2, which are relatively similar in their structure. There is a significant interaction effect between the factors approach and task number ($p = 0.02$, determined using two-way *analysis of variance (*ANOVA)). However, this is not relevant for the interpretation of the two main effects, as the interaction was ordinal (the two lines in Fig. 8 show the same trend).

When comparing the achieved skill sequences of the two

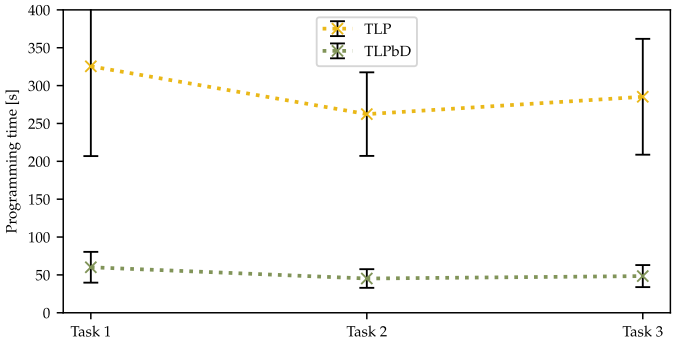

Fig. 8. The plot shows the mean task programming times and standard deviations. TLPbD (green) is about five times faster than TLP (orange).

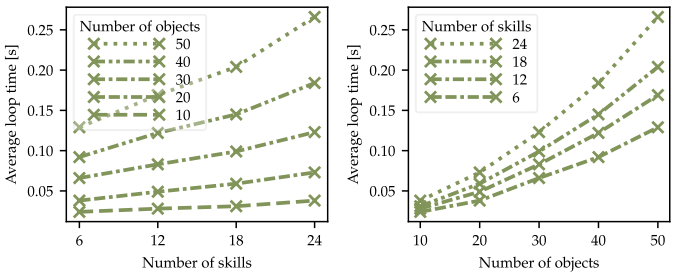

Fig. 9. The two plots show the average loop time of the semantic skill recognizer as a function of the number of skills (left) and objects (right).

programming approaches, the results of TLP showed more errors. Many users did not make use of the Stack and Peg-in-hole skill for task 1 and 2, respectively, but instead used the Place skill. This led to tasks being less generic and fault tolerant in the execution. In case of TLPbD, the system decided on the proper choice of skills. The only error that occurred here was in task 2: When users released the disc above the rod (and not slid onto it), the skill recognizer utilized the Stack instead of the Peg-in-hole skill.

The oral feedback of the participants revealed a clear preference for TLPbD. Despite the system choosing the skills, the participants felt to have the "decision-making sovereignty". They stated that they would require less reading and less knowledge regarding the skills. They described the process as "one flow", as it needed fewer clicks and "less fiddling" compared to TLP.

C. Scalability

In another test, we evaluated the scalability of the system. The user study made use of six skills with in total ten objects. To check if the system can handle more skills and objects in the world, we increased both numbers and measured the average loop time of the semantic skill recognizer. The results are plotted in Fig. 9. The test was run on a basic office computer on six cores (the calculations for each skill can run in a separate process). It can be seen that the loop time is increases linearly with the number of skills, and polynomial (2^{nd} order) with the number of objects.

V. Discussion

Both the results of the two questionnaires and the programming time support our hypothesis that TLPbD substantially outperforms TLP on all scales regarding the intuitive use and the workload. This is also corroborated by the oral feedback. With a time reduction by a factor of five and absolute programming times of less than a minute, the new approach is suitable for automating processes with lot sizes of just a few items. The valuable feedback received during the user study could be used to improve the task-level interface substantially, making its usage more intuitive and maybe faster. However, we highly doubt that this would make the TLP approach as intuitive and fast as TLPbD.

Furthermore, TLPbD does not require tech-savvy users and is less error-prone. In contrast to TLP, the (perceived) technical expertise has only a small influence on the performance. Together with the lower mental load and frustration, this reduces the demands on the robotics expertise of the shop-floor workers. With process knowledge encoded in the PDDL annotations of the skills, the semantic skill recognizer was able to select more appropriate skills than the average human operator did (e. g. Stack instead of Place; the consequences of this are described below). This leads to programs being more reliable under changing conditions.

What makes the approach especially advantageous is that it has been incorporated into the existing TLP interface. Therefore, it inherits all possibilities of the current system, such as editing or reordering of skills after programming. Skills can still be added to a task using the TLP approach, if it might not be possible with TLPbD. This is the case for skills not requiring the robot, such as a simple wait skill.

One limitation of the approach is the restricted observability of the world. The world observer (Section III-B) requires an accurate model of the initial world state and can only register changes if the gripper properly grasps and releases objects. If the robot slides or tilts an object during a motion, this cannot be registered. This would require a physical simulation of the world, as it is suggested in [29]. However, changes in the world triggered by a human worker, for instance, would still not be reflected in the world model. Therefore, camera-based scene recognition is one of our current aims.

A critical aspect of the semantic skill recognition is its reliability in the detection of skills. We have shown that the system can identify six different skills. Yet, these are only a subset of what one would require for a full-scale industrial scenario (Bøgh et al. define 15 essential industrial skills of which we provide 3 [10]). However, the used skills Place, Stack and Peg-in-hole are similar (which can be seen in the choice of skills of the user) and can still clearly be discriminated by the system. There was only one failure case, in which the recognizer chose Stack instead of Peg-in-hole, see Section IV-B. Therefore, we do not expect that adding further skills will influence the success rate of the skill recognizer.

An important requirement for skills to be recognized by the system is that their conditions and effects are observable. Yet, this does not prevent the user from adding the skill manually, as mentioned above. There are also situations, in which a skill can be identified, but not all of its parameters, as these might

not be fully observable. In this case, the correct skill can still be added, but the user needs to provide further information using the TLP interface. Nevertheless, this saves time for the user. The only source of error that has occurred so far is when the robot is moved too fast, compared to the loop rate of the recognizer.

In theory, it is possible that two skills fulfill all their conditions and effects at the same time. At the moment, the semantics of our skills used within the user study do not allow for such a situation and the skill recognizer would not be able to handle this properly. In the future, with more skill being added to the system, such a constellation could possibly occur. For this, we plan to extend the system and query the user to select the desired skill.

Of course, for TLPbD to work, an expert needs to provide the relevant skills, as well as appropriate semantic descriptions. But TLP approaches also require the definition of relevant skills. And in our experience, once the definition of a skill's execution has been programmed, the additional effort required for the PDDL description is negligible.

During the user study, the programmed tasks were not executed. However, we programmed and executed task 1 in another test to proof the effectiveness and capability of the whole chain from demonstration to execution. This is also presented in the video accompanying this paper. It is shown that the execution of the programmed task is successful and the robot stacks the three boxes autonomously. Furthermore, the advantage of using a Stack skill compared to a Place skill is highlighted: It allows for a box to be moved, while further boxes are still stacked on top of it instead of the box' original position. For this, the `ParameterDeducers` convert the recorded absolute poses into relative poses. During the execution, the current poses of the objects are queried from the world model. Details on this conversion and execution are found in [3].

The results of Section IV-C show that the recognizer loop becomes slower with an increasing number of skills and objects. When assuming the aforementioned 15 essential skills [10], and typical assembly tasks with less than 30 items, a single loop in the current implementation would take less than 100 ms. Through code optimization and using a compiled language (rather than Python), it may well be sped up by a factor of 100.

VI. Conclusion

This paper has introduced *Task-Level Programming by Demonstration (*TLPbD), a novel intuitive programming approach which combines *Programming by Demonstration (*PbD) and *Task-Level Programming (*TLP). In TLPbD, users demonstrate tasks purely through PbD, and parameterized skill sequences are automatical identified on-line with a *semantic skill recognition* algorithm. The resulting skill sequences are sent to a TLP interface, where they can be further modified. As the results of a user study confirm, TLPbD is as fast and intuitive as PbD and yet as reliable and easy to use as TLP. Thus, we believe that TLPbD is highly suitable for automating processes with *collaborative robots (*cobots) in *small and medium-sized enterprise (*SME) with low batch sizes.

References

[1] Franka Emika GmbH. Franka Desk. Accessed: 2019-02-20. [Online]. Available: https://www.franka.de/panda/

[2] ArtiMinds Robotics GmbH. Artiminds RPS. Accessed: 2019-02-20. [Online]. Available: https://www.artiminds.com/?page_id=2274

[3] F. Steinmetz, A. Wollschläger, and R. Weitschat, "RAZER–a HRI for visual task-level programming and intuitive skill parametrization," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, 2018.

[4] S. Calinon, "Learning from Demonstration (Programming by Demonstration)," *Encyclopedia of Robotics*, 2018.

[5] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical Movement Primitives: Learning attractor models for motor behaviors," *Neural computation*, vol. 25, no. 2, 2013.

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

[7] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, "Probabilistic movement primitives," in *Advances in NIPS*, 2013.

[8] R. Lioutikov, G. Neumann, G. Maeda, and J. Peters, "Learning movement primitive libraries through probabilistic segmentation," *The International Journal of Robotics Research*, vol. 36, no. 8, 2017.

[9] F. Meier, E. Theodorou, F. Stulp, and S. Schaal, "Movement segmentation using a primitive library," in *IEEE/RSJ IROS*, 2011.

[10] S. Bøgh, O. S. Nielsen, M. R. Pedersen, V. Krüger, and O. Madsen, "Does your robot have skills?" in *43rd Int. Symp. on Robotics*, 2012.

[11] F. Dai, A. Wahrburg, B. Matthias, and H. Ding, "Robot assembly skills based on compliant motion," in *Proc. 47th Int. Symp. Robotics*, 2016.

[12] R. H. Andersen, T. Solund, and J. Hallam, "Definition and initial case-based evaluation of hardware-independent robot skills for industrial robotic co-workers," in *Proc. 41th Int. Symp. Robotics*, 2014.

[13] F. Steinmetz and R. Weitschat, "Skill parametrization approaches and skill architecture for human-robot interaction," in *IEEE Int. Conf. Automation Science and Engineering*, 2016.

[14] C. Paxton, F. Jonathan, A. Hundt, B. Mutlu, and G. D. Hager, "Evaluating methods for end-user creation of robot task plans," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2018.

[15] A. Perzylo, N. Somani, S. Profanter, I. Kessler, M. Rickert, and A. Knoll, "Intuitive instruction of industrial robots: Semantic process descriptions for small lot production," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2016.

[16] J. M. R. C. et al., "A study on the suitability of visual languages for non-expert robot programmers," *IEEE Access*, 2019.

[17] S. Profanter, A. Perzylo, N. Somani, M. Rickert, and A. Knoll, "Analysis and semantic modeling of modality preferences in industrial human-robot interaction," in *RSJ IROS*, 2015.

[18] M. Wächter, S. Schulz, T. Asfour, E. Aksoy, F. Wörgötter, and R. Dillmann, "Action sequence reproduction based on automatic segmentation and object-action complexes," in *Int. Conf. Humanoid Robots*, 2013.

[19] M. Stenmark and E. A. Topp, "From demonstrations to skills for high-level programming of industrial robots," in *AAAI Fall Symposium*, 2016.

[20] J. R. Chen and B. J. McCarragher, "Programming by demonstration - constructing task level plans in hybrid dynamic framework," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 2, 2000.

[21] S. Ekvall and D. Kragic, "Robot learning from demonstration: a task-level planning approach," *Int. Journal of Advanced Robotic Systems*, vol. 5, no. 3, 2008.

[22] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL—the planning domain definition language," Yale Center for Computational Vision and Control, Tech. Rep. CVC TR-98-003/DCS TR-1165, 1998.

[23] S. Brunner, F. Steinmetz, R. Belder, and A. Dömel, "RAFCON: A graphical tool for engineering complex, robotic tasks," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2016.

[24] M. Fox and D. Long, "PDDL2.1: An extension to PDDL for expressing temporal planning domains," *JAIR*, vol. 20, 2003.

[25] K. Karrer, C. Glaser, C. Clemens, and C. Bruder, "Technikaffinität erfassen–der Fragebogen TA-EG," *Der Mensch im Mittelpunkt technischer Systeme*, vol. 8, 2009.

[26] J. Hurtienne and A. Naumann, "QUESI—a questionnaire for measuring the subjective consequences of intuitive use," *Interdisciplinary College*, vol. 536, 2010.

[27] A. Naumann and J. Hurtienne, "Benchmarks for intuitive interaction with mobile devices," in *Proc. 12th Int. Conf. Human-Computer Interaction with Mobile Devices and Services*, 2010.

[28] S. G. Hart and L. E. Staveland, "Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research," in *Human Mental Workload*, ser. Advances in Psychology, 1988, vol. 52.

[29] A. S. Bauer, P. Schmaus, A. Albu-Schäffer, and D. Leidner, "Inferring semantic state transitions during telerobotic manipulation," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2018.