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

Collaborative Programming of Conditional Robot Tasks

Christoph Willibald, Thomas Eiband, and Dongheui Lee

***Abstract*— Conventional robot programming methods are not suited for non-experts to intuitively teach robots new tasks. For this reason, the potential of collaborative robots for production cannot yet be fully exploited. In this work, we propose an active learning framework, in which the robot and the user collaborate to incrementally program a complex task. Starting with a basic model, the robot's task knowledge can be extended over time if new situations require additional skills. An on-line anomaly detection algorithm therefore automatically identifies new situations during task execution by monitoring the deviation between measured- and commanded sensor values. The robot then triggers a teaching phase, in which the user decides to either refine an existing skill or demonstrate a new skill. The different skills of a task are encoded in separate probabilistic models and structured in a high-level graph, guaranteeing robust execution and successful transition between skills. In the experiments, our approach is compared to two state-of-the-art Programming by Demonstration frameworks on a real system. Increased intuitiveness and task performance of the method can be shown, allowing shop-floor workers to program industrial tasks with our framework.**

I. Introduction

Traditional robot programming methods used in today's production rely on expert knowledge and advanced programming skills [1]. Due to the lack of flexibility and intuitiveness of these programming methods, non-experts such as shop floor workers have difficulties to instruct robots for new tasks. Favored by the development of new generations of collaborative robots [2], however, users can easily guide a robot along a desired trajectory and thereby demonstrate different motions. In contrast to typical repetitive sequential robot tasks in the production, conditional tasks involve a decision about how to react to different observations. As this requires a state, where the robot decides on how to proceed, this has been conventionally achieved by writing code, such as an *if*-statement. Instead, we facilitate to program such functionality by demonstration utilizing the robot's sensing capabilities. A robot is then not only capable of reproducing a demonstration, but can also automatically distinguish between several different situations and react accordingly.

In Programming by Demonstration (PbD), a state-action mapping is learned from demonstrations provided by a user. Two key questions of recent research on active learning [3] [4] are when to demonstrate additional behavior and on what regions of the input space to focus on with these demonstrations. While other PbD approaches rely on the user to monitor the task reproduction in order to identify new situations and task deficiencies [5]–[8], our proposed active learning framework automatically detects anomalies during the execution. This triggers a teaching phase in which the user incrementally refines or extends the robot's task knowledge, leading to higher autonomy when facing similar conditions in the future.

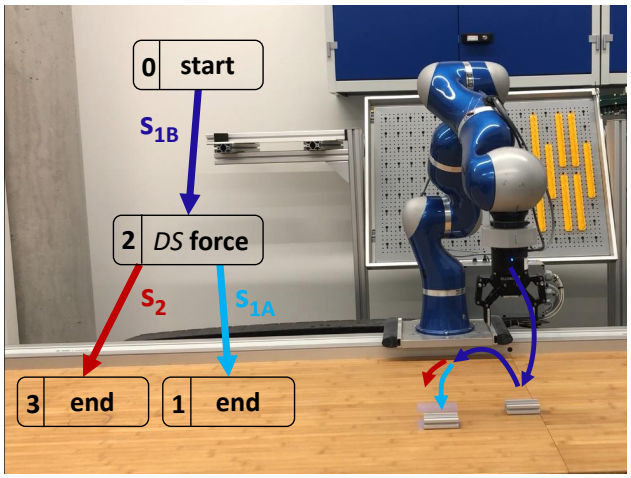


Fig. 1. Interactive PbD framework for programming of conditions, represented as decision states (DS) of a task-representing graph. In this example, a sensed contact force triggers a decision to execute either skill s_{1A} or s_2.

The contribution of this work is a collaborative robot-programming framework to incrementally generate and refine a graph that structures the probabilistically encoded skills of a task on a high level. The interactive teaching process not only reduces the workload of the user but also prevents potentially dangerous situations by detecting anomalies at an early stage. The topological arrangement of the skills and the explicit programming of *decision states*, as shown in Fig. 1, eliminates perceptual aliasing and guarantees a successful transition between skills when reproducing known situations. That allows to scale the complexity of a task over time without reducing the overall performance. The low level statistical modeling of the skills compensates local perturbations in the environment and the model inherent probability distribution can be used to detect deviations from the intended behavior.

By combining kinesthetic teaching with a graphical user interface (GUI) that guides the user throughout the entire teaching process, we present an intuitive interface that maximally supports novice users. As shown in [9], the additional feedback loop through combination of execution and teaching, closes the gap between the user's assumed and the robot's actual task knowledge, which further improves the teaching performance.

C. Willibald, T. Eiband and D. Lee are with the German Aerospace Center (DLR), Institute of Robotics and Mechatronics, Wessling, 82234, Germany
D. Lee is also with the Technical University of Munich, Chair of Human-centered Assistive Robotics, Munich, 80333, Germany
firstname.lastname@dlr.de
This work has been partially supported by Helmholtz Association.

II. Related Work

Common methods for encoding a behavior require multiple user demonstrations of the same skill under slightly different environment conditions [6] [8] [10]. However, with the proposed framework we reduce the number of initial skill demonstrations. A key question in PbD is how to encode a task in order to robustly reproduce the demonstrated behavior, while measuring the robot's imitation performance. Current approaches differentiate between a low-level *trajectories encoding* and a high-level *symbolic encoding*. By monitoring and comparing the user's task demonstrations to pre- or post-conditions of a predefined behavior set, a symbolic encoding partitions the task in action segments [11] [12]. In [8] this step is followed by arranging the segments of different task demonstrations in a generalized topology, while in [12], a recognized sequence of parameterized skills is propagated to a graphical interface for further processing by the user. The obtained sequence of skills is forwarded to a symbolic planner for execution. A commercially available example for a high level graphical programming scheme is Franka Desk [13], that allows a user to combine and parameterize different skills for execution on a robot. In contrast to our approach, a task programmed with the mentioned methods is composed of a combination of predefined skills.

Trajectories encoding takes place on a low level of the motion and tries to directly retrieve a non-linear mapping between sensory observation and motor action. For this, a large body of work uses skill encoding based on statistical modeling [10] [14] [15], dynamical systems [16] [17], or a combination of both [18] [19]. The algorithm developed in [18] extracts task constraints from a probabilistic model, used to modulate a dynamical system. Dynamic Movement Primitives (DMPs) [20] are a well-known approach to guarantee convergence to a goal state, using nonlinear dynamical systems and non-parametric regression. In [16], a robot learns an initial policy for a pool stroke in the form of a DMP through PbD, which is autonomously optimized by the robot with a reinforcement learning algorithm. This requires the design of a task specific representation as well as a task specific cost function, whereas our approach is task independent and can be used without additional parameter tuning. Pastor et al. [17] realize early failure detection and online movement adaption with Associative Skill Memories (ASM) that combine DMPs with stereotypical sensor traces.

Other approaches encode a task based on statistical modeling to capture the high variability between demonstrations. Kulić et al. [15] use Hidden Markov Models to reproduce and recognize repeated motion primitives. The frameworks presented in [10] and [14] use time parameterized Gaussian Mixture Models (GMM) to probabilistically encode the demonstrations and Gaussian Mixture Regression (GMR) to condition the multivariate normal distributions on a time vector. Eiband et al. [10] present a method, that allows active transitioning between skills of a solution pool when anomalies are detected during task execution. In our framework, the skills are structured in a graph, leading to a smoother and more robust skill transition.

In [7], [6] and [21], low-level trajectory encodings such as DMPs are topologically structured on a high task level. Kappler et al. [7] present a data-driven approach which structures ASMs in a *manipulation graph* and triggers a skill transition during task execution, based on high level sensory feedback. The considered successor ASMs are determined by the *manipulation graph*. In contrast to our approach, the user needs to manually construct the initial *manipulation graph* by choosing predefined skills from an ASM library. Niekum et al. [6] present an approach in which user demonstrations are segmented into reusable skills, used to create a finite state automaton (FSA). If the user intervenes during the execution to provide corrective demonstrations, the pool of task demonstrations is again segmented, limiting reusability of previously learned FSAs. In [21], a high level behavioral tree hierarchically decomposes the task into sub-tasks. This enables a cooperative execution, in which sub-tasks can be assigned either to a human or a robot.

An incremental learning framework is proposed in [5], where the user physically guides the robot, creating a sequence of coherent states that capture the demonstrated configurations in a *Robot State Automaton* (RSA). While demonstrating, the user has to identify situations, in which the task splits into different possible skills. When the user inserts a branching state in the RSA, a camera image is recorded and programmed as the condition for a transition to the subsequently demonstrated state. We argue that manually programming the transition conditions and creating branching states in which the robot decides between different skills is not intuitive for the user, especially if multi-modal sensor values are considered for a transition. This might lead to erroneous decisions in branching states, which can be avoided by our approach that automatically detects new situations during execution and collaborates with the user to provide alternative skills.

While previously described methods [5] - [8] rely on the user to identify unintended or new situations, the approaches in [3], [4], [10] and [16] aim to automatically detect anomalies or task deficiencies. Maeda et al. [3] propose a method based on Gaussian Processes to quantify the generalization capability of an extrapolated trajectory for an unseen situation. In [4], a mapping from observation to action, bound on a finite set of basic actions, is learned with a GMM. The GMM provides a classification confidence with which an observation can be assigned to one of the action-clusters. The task outcome prediction of [16] compares sensor experience recorded during multiple successful task executions with the perceived sensor values. A z-test, determines if the current execution is correlated with the successful examples. In contrast to that, in [10], the sequence of mean vectors and covariance matrices of the probabilistic skill encoding is used in every time step to compute a Mahalanobis distance, quantifying the deviation between measured and commanded state. Each of the above mentioned frameworks includes either a manual action-labeling or error threshold tuning. However, we propose an unsupervised approach that learns an anomaly boundary from previous experience.

III. Interactive Teaching Procedure

With our proposed interactive teaching process, a user can incrementally program a robot by adding new skills or

refining existing skills of a task model when the robot has identified a new situation. In the beginning, the robot does not have any task knowledge, so a task-representing graph is built from scratch. The teaching process starts with an initial user demonstration. After the user has demonstrated the first skill, s/he restores the environment, i.e. manually putting back all objects to their initial locations. To increase the variability of the training data, especially in the force domain during object interaction, the robot repeats the movement and records the sensor readings without the user's guidance. These two sensor data sequences from user demonstration and robot execution are used to encode the skill in a probabilistic model (see Sec. IV-A). The resulting trajectory of mean vectors $\boldsymbol{\mu_t}$ and covariance matrices $\boldsymbol{\Sigma_t}$, represents the first skill and is appended to the start state of the task-graph. The robot is now capable of reproducing the task, which currently consists of only one skill, and starts the execution. During the execution phase, an on-line anomaly detection algorithm constantly monitors the deviation between commanded and recorded sensor values (see Sec. IV-B). As seen in Fig. 2, the robot stops the movement when an anomaly is detected and interacts with the user to resolve the new situation. If the user chooses to ignore the anomaly, the robot continues with the task execution and does not consider it as a new situation. In case the user wants to refine the current skill, s/he can do so via *user-refine* or *auto-refine* mode (see Sec. IV-C *(b)*). If the user

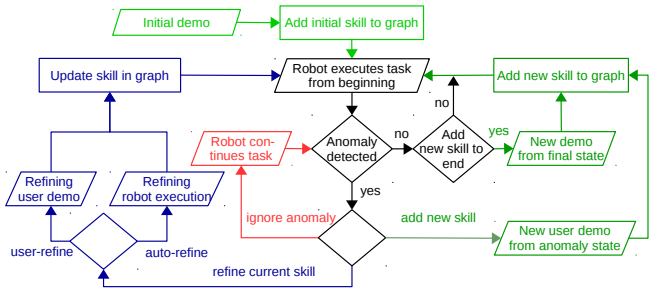


Fig. 2. Combined execution and teaching phase, in which the robot and the user collaborate to solve new situations. The user can ignore an anomaly (red), add new skills (green) or refine existing skills (blue).

decides to add a new skill to the graph after an anomaly was detected, s/he needs to demonstrate a behavior that resolves this situation in future executions. The further procedure is analog to the initial skill demonstration described above (see also Sec. IV-C *(a)*).

After the robot has successfully executed the complete task, the user can also extend the graph by giving a new demonstration starting from the current end-state configuration. The procedure is similar to demonstrating a new behavior from an anomaly, with the only difference that the demonstrated behavior is instead appended to the end-state.

IV. Graph-Based Conditional Task Learning

A. Probabilistic Encoding of Demonstrations

As mentioned, with our approach, the user only needs to demonstrate a new behavior once, in order to teach a new skill. To incorporate the robot's dynamics of the execution without the user's guidance, the robot repeats the trajectory of the user demonstration and records a second sequence. The different sensor recordings of the task's two examples are used to extract important features, in which the robot must be exact during reproduction. In parts with more variability, however, higher deviations are accepted during the execution, which increases the overall robustness. Only proprioceptive measurements from the robot's sensors are considered. Particularly no vision based monitoring of the environment is used, which is sufficient for the partially structured production environment and eliminates errors resulting from poor visibility. The obtained vector of sensor data $\boldsymbol{x}_t = [\boldsymbol{p}, \boldsymbol{o}, \boldsymbol{f}, \boldsymbol{t}, g, h]^T \in \mathbb{R}^{15}$ at time t, consists of the end-effector's Cartesian position $\boldsymbol{p} = [x, y, z]$, orientation in unit quaternions $\boldsymbol{o} = [q_w, q_x, q_y, q_z]$, force $\boldsymbol{f} = [f_x, f_y, f_z]$ and torque $\boldsymbol{t} = [t_x, t_y, t_z]$, as well as the gripper finger distance g and the grasp status $h \in \{-1, 0, 1\}$, depicting "no grasp", "gripper moving" and "grasping" respectively. The sensor values are recorded with a frequency of $f_{\text{rec}} = 1000$ Hz and are down-sampled to $f_{\text{ds}} = 50$ Hz for a more efficient data handling. The training data is stored in matrix $\boldsymbol{X}_{\text{U}} = [\boldsymbol{x}_{\text{U},1}, ..., \boldsymbol{x}_{\text{U},N_{\text{U}}}] \in \mathbb{R}^{15 \times N_{\text{U}}}$ for the user demonstration and $\boldsymbol{X}_{\text{R}} = [\boldsymbol{x}_{\text{R},1}, ..., \boldsymbol{x}_{\text{R},N_{\text{R}}}] \in \mathbb{R}^{15 \times N_{\text{R}}}$ for the robot demonstration with respective sample length N_{U} and N_{R}. Similar to [10], we first apply dynamic time warping (DTW) to align the two sequences on a common time axis and equalize their length N. In order to make sure that all dimensions contribute equally to the warping error, the data is standardized by subtracting the mean and dividing by the standard deviation in every dimension (z-transformation). For the next step, we undo the standardization by applying the inverse z-transformation on every dimension of the aligned sensor sequences and learn a time-based GMM to generalize over the demonstrations. An expectation-maximization (EM) algorithm is used to estimate the multivariate Gaussian distribution for the input matrix

$$\boldsymbol{G}_s = \begin{bmatrix} \boldsymbol{n} & \boldsymbol{n} \\ \boldsymbol{X}_{\text{U}} & \boldsymbol{X}_{\text{R}} \end{bmatrix} \in \mathbb{R}^{16 \times 2N} \tag{1}$$

of a new skill s. The EM algorithm is initialized using k-means clustering and setting the number of clusters proportional to the length N of the demonstrated time series to $k = \frac{N}{f_{\text{ds}}}$. The obtained model $\mathcal{M} = GMM(\boldsymbol{G}_s)$ and the time vector $\boldsymbol{n} = [1, ..., N]$ serve as input to the GMR in order to compute a regression $GMR(\mathcal{M}|n)$ over all dimensions. The GMR provides a generalized trajectory of sensor values $\boldsymbol{Y}_s = [\boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_N] \in \mathbb{R}^{15 \times N}$ with an associated sequence of covariance matrices $\boldsymbol{Z}_s = [\boldsymbol{\Sigma}_1, ..., \boldsymbol{\Sigma}_N] \in \mathbb{R}^{15 \times 15 \times N}$. The obtained mean value sequence together with the covariance matrices are required for a monitored execution of the skill.

B. On-Line Anomaly Detection

During the execution of a probabilistically encoded skill, the robot can autonomously detect new situations. For every sensor modality a real-time monitoring component constantly compares the commanded and measured values to detect abnormal deviations. We are therefore not only able to detect anomalies with our one-class classification algorithm

on-line, but can also assign them to a sensor category. According to the modalities, the different categories are: end-effector position/orientation, force/torque, as well as gripper opening and grasp status. This enables the anomaly detection algorithm to identify new situations that result from missing or shifted objects, different object geometries or weights, only with the robot's proprioceptive sensors, including force measurements. For every timestep t of the execution, the deviation between the measured state $\boldsymbol{m}_{i,t}$ and commanded state $\boldsymbol{\mu}_{i,t}$ of a sensor modality i is quantified using the Mahalanobis distance metric

$$D_{\mathrm{M}(i,t)} = \sqrt{(\boldsymbol{m}_{i,t} - \boldsymbol{\mu}_{i,t})^T \boldsymbol{\Sigma}_{i,t}^{-1} (\boldsymbol{m}_{i,t} - \boldsymbol{\mu}_{i,t})} \quad . \tag{2}$$

In combination with a custom anomaly threshold ϵ_i for every modality of the skill, this metric leads to a higher error sensitivity in timesteps where the execution needs to be precise, indicated by small values of the reduced covariance matrix $\boldsymbol{\Sigma}_{i,t}$. If any $D_{\mathrm{M},i}$ exceeds its skill and modality specific anomaly threshold ϵ_i for e consecutive timesteps, an anomaly is detected (see Fig. 3a). In contrast to comparable state of the art methods, our approach does not rely on manual error threshold tuning. Instead, we compute an independent anomaly threshold ϵ_i for every modality of a skill, based on the recorded data of the user demonstration U and robot sampling R. With (3) and (4), we determine the highest occurring Mahalanobis distance for deviations between the sensor values $\boldsymbol{m}_{d,i,t}$ of all demonstrations $d \in \{\mathrm{U}, \mathrm{R}\}$ belonging to one skill and the associated mean $\boldsymbol{\mu}_{i,t}$ of the same skill. This process is repeated for every sensor modality of a skill.

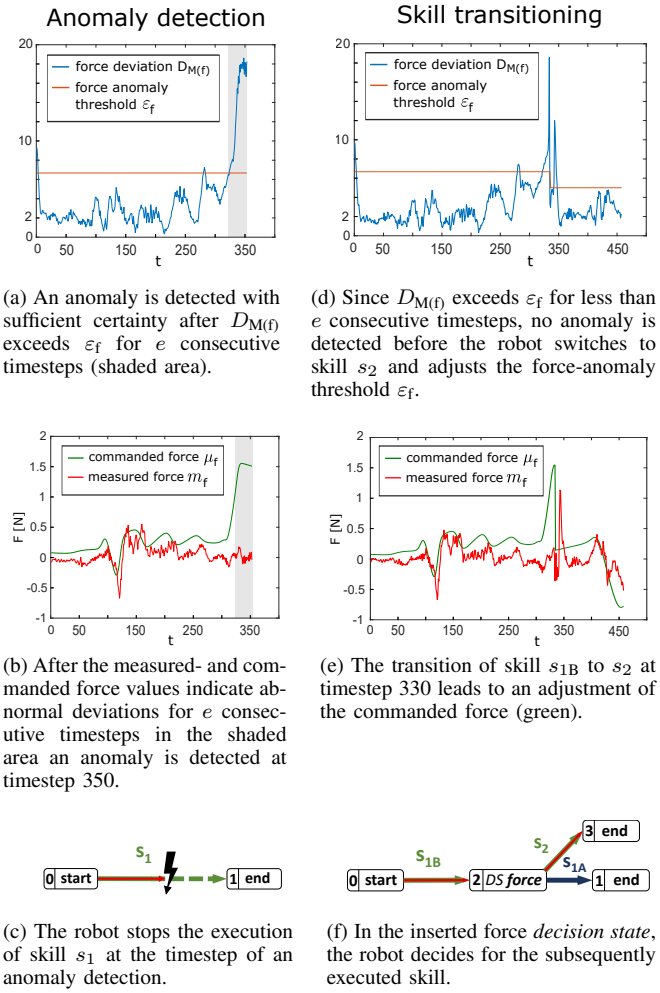
$$\tilde{D}_{\mathrm{M}(d,i)} = \max_{t \in [1,N]} \sqrt{(\boldsymbol{m}_{d,i,t} - \boldsymbol{\mu}_{i,t})^T \boldsymbol{\Sigma}_{i,t}^{-1} (\boldsymbol{m}_{d,i,t} - \boldsymbol{\mu}_{i,t})} \tag{3}$$

$$\epsilon_i = \max_{d \in \{\mathrm{U},\mathrm{R}\}} \tilde{D}_{\mathrm{M}(d,i)} \tag{4}$$

C. Incremental Graph Generation and Refinement

The robot gains additional task knowledge from the user demonstrations over time and incrementally builds a graph, structuring the skills on an abstract task level (see Fig. 3c and Fig. 3f). The demonstrated skills, represented by the graph's edges, are connected by nodes, termed *decision states*. The states restrict the number of possible successor edges and handle the transition between skills, described in more detail in Sec. IV-D. A teaching phase, in which changes are made to the graph, is always triggered by the robot if a new situation is detected during the execution. If an anomaly is detected, the robot stops the current skill execution at t_{anomaly} (see Fig. 4a). This can be caused, for example, by an unexpected object weight during a pick action, indicated by a deviation in the force-torque sensor values. It is then within the user's responsibility to decide whether the detected situation *a)* poses a new skill or *b)* must be incorporated into the currently executed skill.

a) Extend Graph With a New Skill: If the user decides, that the newly detected situation requires an additional skill, s/he demonstrates a new behavior, starting from the robot configuration in t_{anomaly} (see x_i in Fig. 4b). This skill handles the newly detected situation in future executions of



(a) An anomaly is detected with sufficient certainty after $D_{\mathrm{M(f)}}$ exceeds ε_{f} for e consecutive timesteps (shaded area).

(d) Since $D_{\mathrm{M(f)}}$ exceeds ε_{f} for less than e consecutive timesteps, no anomaly is detected before the robot switches to skill s_2 and adjusts the force-anomaly threshold ε_{f}.

(b) After the measured- and commanded force values indicate abnormal deviations for e consecutive timesteps in the shaded area an anomaly is detected at timestep 350.

(e) The transition of skill $s_{1\mathrm{B}}$ to s_2 at timestep 330 leads to an adjustment of the commanded force (green).

(c) The robot stops the execution of skill s_1 at the timestep of an anomaly detection.

(f) In the inserted force *decision state*, the robot decides for the subsequently executed skill.

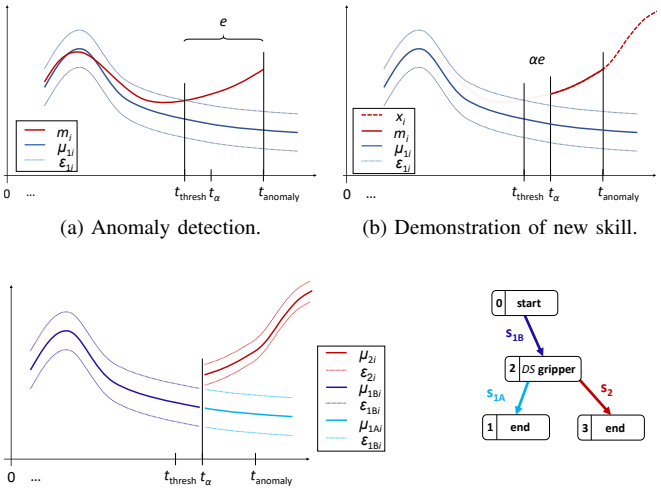
Fig. 3. Anomaly detection (left column) and skill transition when reproducing the situation with the extended task graph (right column). The green arrows in the graph represent the selected skills and the red arrows the progress of the execution.

the task. The sensor trajectory of the user demonstration $\boldsymbol{X}_{\mathrm{Udem}}(t_{\mathrm{anomaly}}, ..., N_{\mathrm{U}})$ is appended to the recorded sensor sequence $\boldsymbol{M}(t_\alpha, ..., t_{\mathrm{anomaly}})$ during the interval $[t_\alpha; t_{\mathrm{anomaly}}]$, resulting in $\boldsymbol{X}_{\mathrm{U}}$. After the user has brought the environment to the state before the demonstration, the robot moves to the configuration at timestep t_α and repeats the extended user demonstration $\boldsymbol{X}_{\mathrm{U}}$. The two recorded sensor sequences of the alternative behavior are then probabilistically encoded and saved as skill s_2. A new *decision state* is introduced in the task-graph, splitting up skill s_1 into two parts before and after the anomaly detection, depicted $s_{1\mathrm{B}}$ and $s_{1\mathrm{A}}$ respectively (see Fig. 4c and Fig. 4d). Additional to $s_{1\mathrm{A}}$, s_2 is appended to the newly inserted node. As seen in Fig. 4c, skill s_1 is split at timestep

$$t_\alpha = t_{\mathrm{thresh}} + \alpha e \ , \tag{5}$$

where t_{thresh} is the timestep in which the sensor deviation $D_{\mathrm{M}(i,t)}$ first exceeds the anomaly threshold ϵ_i, e is the number of consecutive timesteps for which $D_{\mathrm{M}(i,t)} > \epsilon_i$ until an anomaly is triggered and $\alpha \in [0; 1[$ is a scaling factor, placing the *decision state* in between timestep t_{thresh} and

t_{anomaly}. An early and smooth transition from skill $s_{1\text{B}}$ to its successor without unnecessarily long anomaly reproduction, requires a minimal α, placing the *decision state* close to t_{thresh}. However, a robust decision requires distinguishable sensor readings that can be assigned to a specific skill with a certain confidence, pushing the *decision state* towards t_{anomaly} and thereby $\alpha \to 1$. The decision for the subsequent skill must be made before t_{anomaly} is reached during execution of skill $s_{1\text{B}}$, otherwise the anomaly detection wrongly identifies a new situation for the case s_2 (see Fig. 3d). Preliminary experiments have shown that setting $e = 30$ and $\alpha = 1/3$ is a good compromise between robustness and acceptable decision-making delay.



(a) Anomaly detection. (b) Demonstration of new skill.

(c) Splitting up skill s_1 at t_α. (d) Updated task-graph.

Fig. 4. The process of adding a new skill to the task-representing graph.

b) Refine Current Skill: In case the user wants to refine the skill s, during which the anomaly was detected, its generalized sensor value trajectory $\boldsymbol{Y}_s$ with associated sequence of covariance matrices $\boldsymbol{Z}_s$ is adjusted. Hereby, no new skill is generated, but the current one becomes capable of handling more diverse conditions. That enables the robot to account for deviations in the environment during the task execution, such as different object weights or geometries, which must be handled identically and therefore require the execution of the same skill. For example, a geometry-based sorting task of different objects, in which the weight of an object shall not have an influence on the task outcome, can be realized by refining an initial demonstration with further examples. The first appearance of an object with a different weight triggers a force anomaly, which should be regarded as an expected condition in future executions of that skill. By incorporating a demonstration of the new situation into the probabilistic skill model, the anomaly threshold ϵ_i for every sensor modality is adjusted, avoiding false-positive force anomaly detection results in future executions. An example of the new setup is either acquired by a user demonstration in *user-refine* mode or by the robot in *auto-refine* mode. The latter describes the robot continuing the motion after a detected anomaly until the end of the skill and recording the new sensor value sequence $\boldsymbol{X}_{\text{ref}}$. During execution of the remainder of the skill, the anomaly detection is not active. A manual demonstration in *user-refine* mode, starting from the anomaly configuration, offers the possibility to adjust the full trajectory of the correction. For both options, the recorded sequence of sensor values $\boldsymbol{X}_{\text{ref}}(t_{\text{anomaly}}, ..., N_{\text{ref}})$ is appended to the sensor trajectory $\boldsymbol{M}(t_1, ..., t_{\text{anomaly}})$ of the skill before the anomaly. The new refining demonstration is used together with the initial training data of that skill for a new probabilistic encoding, as described in Sec. IV-A. The task-graph is then updated with the new model of the skill.

D. Execution of the Learned Behavior

An essential concept of our active learning framework is the combined teaching and execution in a collaborative way. The task-representing graph enables the robot to robustly reproduce learned behavior and can be refined or extended by the user, when the robot detects a new situation. Structuring the different skills on a high task-level, while using statistical modeling for the skills on a low level, poses multiple advantages. *Decision states* are automatically inserted at critical skill transition timesteps of the task, which accelerates not only the computation required for the decision process, but also eliminates perceptual aliasing and thereby the risk of deciding for a wrong skill. By leveraging the knowledge of an approaching decision, the next skill can be chosen at an early timestep, which avoids unnecessary robot movements. Furthermore, our proposed approach identifies the sensor modality causing an anomaly. That allows to only consider relevant sensor values when deciding for the appropriate subsequent skill in a *decision state*.

As seen on the right side of Fig. 3, the robot starts the task with the first skill $s_{1\text{B}}$. If no anomaly is detected during the execution, the robot reaches the first *decision state*, in which the subsequent skill $s \in \{s_{1\text{A}}, s_2\}$ is determined using the following equation

$$s = \underset{s}{\operatorname{argmin}}(\|\boldsymbol{m} - \boldsymbol{\mu}_{s,0}\|) \ . \tag{6}$$

$\boldsymbol{m}$ is the measured vector of the robot's proprioceptive sensor values of a modality in the *decision state* and $\boldsymbol{\mu}_{s,0}$ is the vector of the first timestep of a generalized sensor trajectory of the same modality. Among all skills attached to the *decision state*, the skill s with the shortest Euclidean distance $\|\boldsymbol{m} - \boldsymbol{\mu}_{s,0}\|$ fulfills the transition condition and is executed next (see Fig. 3e). In contrast to the anomaly detection, we use the Euclidean distance metric here, because the Mahalanobis distance favors skills with high uncertainty in the first timestep, expressed by high values in the covariance matrix.

V. Experiments

We evaluate that our framework is capable of learning tasks that can contain different conditions which need to be observed by the robot during execution. Furthermore, we compare our framework with two alternative approaches that allow intuitive transfer of conditional tasks to a robot.

A. Experimental Setup

As seen in Fig. 5, a DLR LWR IV [2] is mounted on a linear axis and equipped with a "Robotiq 85" 2-finger gripper as well as a FT-sensor, measuring the forces and torques acting on the end-effector. The robot is impedance controlled with a control frequency of 50 Hz and parameterized with constant stiffness- and damping coefficients k_{trans} = 1200 N/m, k_{rot} = 100 Nm/rad and d_{m} = 0.3 Ns/m respectively. Pedals and a tablet displaying a GUI allow the user to

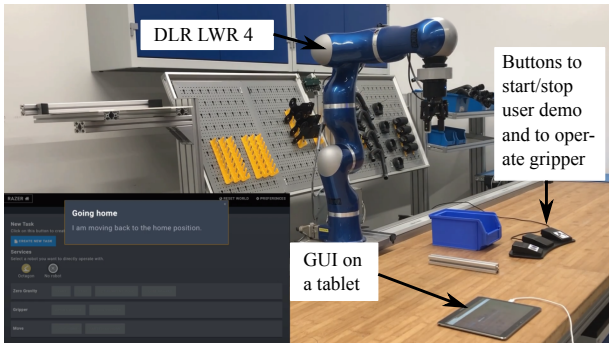


Fig. 5. Experimental setup.

interact with the robot while guiding it at the same time. The pedals are used to open or close the gripper and to start or stop the sensor recording when using kinesthetic teaching in gravity compensation. A GUI guides the user through the teaching process and requests input from the user when the task definition requires it. For all experiments, we use the same object (6.8 cm x 4 cm x 2 cm) in different setups, shown in Fig. 6.

We designed two different tasks; Reorientation and Contact-Based Sorting. The Reorientation task is to manipulate an object from a start to a target. The object's long edge shall be aligned with a mark on the table at the target. In addition, the object can be rotated by 90° in the start location such that the gripper can grasp it over its short edge. This requires a reorientation of the object before placing it in the target location. A step-wise description is shown in Fig. 7 and Fig. 8. The Contact-Based Sorting task is to fill a part storage starting with target I. If target I is occupied, the object shall be placed on target II. The manipulation steps as well as the generation of the task graph are shown in Fig. 10 and Fig. 11.

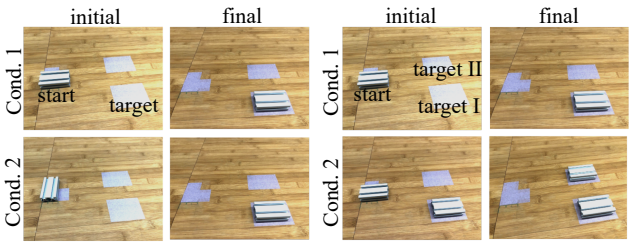


Fig. 6. Initial and final state of the Reorientation (left) and Contact-Based Sorting tasks (right) with each different environmental conditions (Cond. 1 and Cond. 2).

TABLE I
DIFFERENT METHODS COMPARED IN THE EXPERIMENT

Methods / Properties	**Sequential Batch Progr. (SBP)**	**Collaborative Incremental Progr. (CIP)**	**User-Trigger. Incremental Progr. (UIP)**
knowledge representation			
teaching-interaction	unidirectional	bidirectional	unidirectional
incrementally extendable	✘	✔	✔
anomaly detection	robot	robot	user

B. Methods to be Compared

Table I shows an overview of the PbD frameworks we compare, which all use the same sensory input but no vision system. Sequential Batch Programming (SBP) is based on the framework presented in [10], where the teaching and execution are split up in two distinct phases. First, the teacher successively demonstrates all different task solutions, which are independently stored in a solution pool. Whenever an anomaly is detected during the execution of a task solution, the system switches to the state within an alternative solution that minimizes the error between current sensor values and all alternative solution states. Collaborative Incremental Programming (CIP) is our proposed PbD approach that combines anomaly detection with collaborative programming to account for new task conditions. Compared to SBP, the decision state is explicitly programmed by actively querying the user instead of allowing arbitrary switching states, which do not guarantee a successful transition. User-Triggered Incremental Programming (UIP) is inspired by the framework presented in [5], where similar to CIP, a task-representing graph is incrementally constructed in a combined teaching and execution phase. The difference between these methods is, that the teacher has to detect anomalies with UIP during

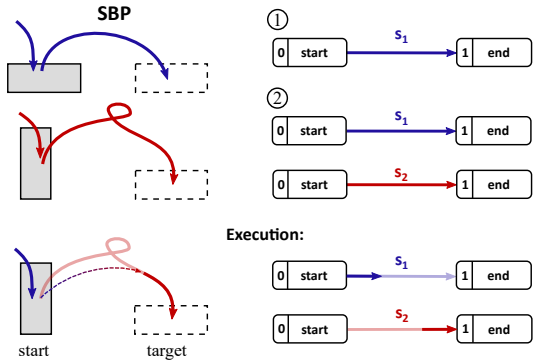


Fig. 7. Task 1 - Reorientation, SBP: In step (1), the user demonstrates a pick and place skill s_1. In step (2), the user extends the solution pool with a second skill s_2, in which the object gets rotated by 90° before placing it in the target location. During execution of the nominal solution s_1, the rotated object in the start location causes an anomaly, that triggers a transition to the alternative solution. The bottom row illustrates an example of a failed execution, where the robot decides for a wrong entry point of the alternative and skips the reorientation part of s_2.

execution of the task and needs to decide if and when a new skill demonstration is needed.

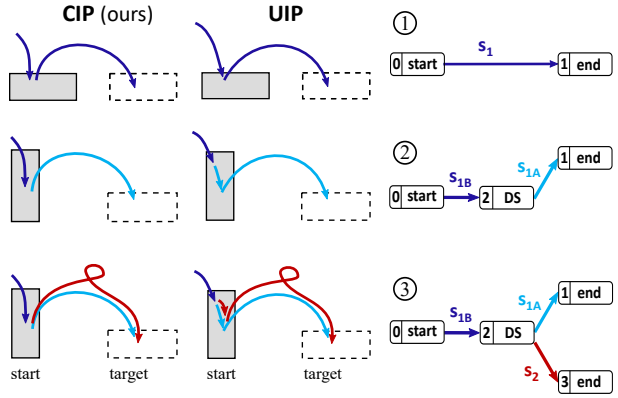
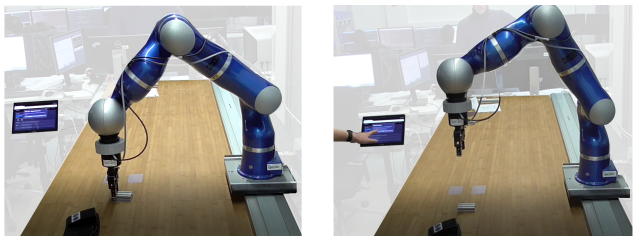


Fig. 8. Task 1 - Reorientation, CIP and UIP: In step (1), the user demonstrates a pick and place skill s_1. Step (2) shows the updated graph after first execution where an anomaly leads to inserting decision state (DS) and splitting s_1 into s_{1A} and s_{1B}. The DS is created by the anomaly detection algorithm in CIP and by the user manually in UIP. In step (3), the user adds a new skill s_2 that accounts for the anomaly and properly rotates the object before placing it.



(a) **CIP**: Autonomous anomaly detection **when** grasping object.

(b) **UIP**: Example for an incorrect user-triggered anomaly **before** grasping object.

Fig. 9. Correct (a) and wrong (b) robot configuration to provide an alternative skill for solving a new situation. Due to the user's influence on the anomaly detection, a configuration in which the robot can't sense the anomaly is more likely with UIP.

The methods were tested by two users that did not work with the system before. They were instructed how to use the GUI and buttons to operate the robot (Fig. 5). They were briefed about the robot's sensing capabilities and that the robot is able to "feel" forces and to measure what is inside the gripper. The users were further advised that the system is not equipped with a vision system.

C. Results

The three methods were compared objectively based on execution success in Table II.

1) Task 1 - Reorientation: When using SBP, a user failed in trial 1 to program the Reorientation task such that during the execution the robot correctly detected an anomaly but switched to a point in the alternative solution that did not reorient the object (Fig.7 bottom). This is a limitation of this method as it does not guarantee feasible solution switching. In the second trial, the task has been programmed successfully by providing an alternative solution with a distinct entry point without further ambiguous states compared to the sensor values at the anomaly. Programming with CIP led to a successful execution in both trials. (Fig. 8 left). With UIP, the decision state has been manually triggered before the robot was actually able to sense an anomaly (Fig. 9b). The task is shown in Fig 8 middle, which caused the execution to fail as there is no grasping anomaly present at the decision state.

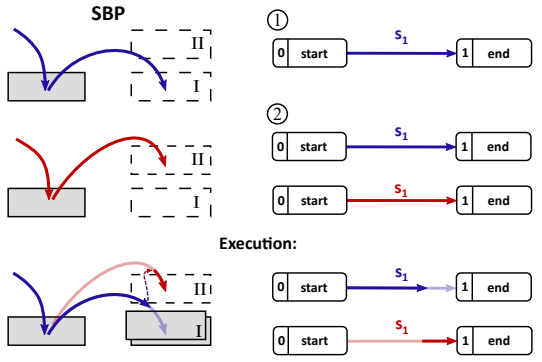


Fig. 10. Task 2 - Contact-Based Sorting, SBP: The user successively demonstrates two pick and place skills in step (1) and (2). In demonstration of skill s_2, the object is placed in target location II, if target location I is occupied by another object. The bottom row shows the execution of the nominal solution s_1, where an unexpected contact force triggers a transition to s_2 while approaching target location I. The robot interpolates to the entry state of the alternative solution and places the object in location II.

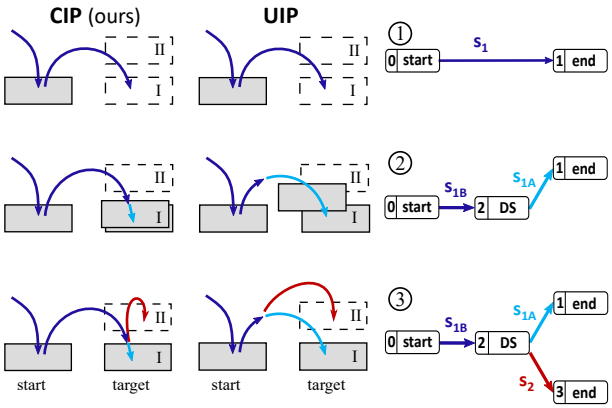


Fig. 11. Task 2 - Contact-Based Sorting, CIP and UIP: Step (1) shows the initial demonstration of a pick and place skill s_1. Step (2) shows the updated graph after first execution where an anomaly leads to decision state (DS) insertion and splitting of s_1 into s_{1A} and s_{1B}. The DS is created by the anomaly detection algorithm in CIP and by the user manually in UIP. In step (3), the user added a new skill s_2 that recovers from the anomaly.

2) Task 2 - Contact-Based Sorting : In this task, SBP and CIP could be used to program a successful task execution. Although no explicit decision point has been programmed in SBP, a feasible path existed from the detected anomaly when contacting an object at occupied target I (Fig. 10) towards the alternative solution that handles target II. With CIP, task definition was successfully handled by the collaborative programming scheme, leading to the task graph shown in Fig. 11. Again, programming with UIP failed as the decision state has been triggered too early by the user. Hereby, the robot could not measure the contact force of an existing object at target I, because the decision state has been inserted wrongly before any contact occurred (Fig. 11 middle).

TABLE II
EXPERIMENTAL RESULTS OF SUCCESSFUL EXECUTIONS

Method	SBP		CIP (ours)		UIP	
Task	1	2	1	2	1	2
Trial 1	✘	✔	✔	✔	✘	✘
Trial 2	✔	✔	✔	✔	✘	✘

D. Discussion

In theory, all three methods allow the user to program the robot by demonstration such that it is able to solve the experimental tasks. With SBP, conditions are not programmed explicitly as it rather reacts to any anomaly instead of making decisions at specified points. Furthermore, each solution needs to be demonstrated over the whole task, whereas in both of the other methods, demonstrations start at possible decision states. With CIP and UIP, conditions are programmed explicitly. The limitation of UIP is that it does not guarantee that a condition can be sensed at the time the user triggers a decision state.

For SBP and UIP, the responsibility for anomaly detection and insertion of decision states lies by the user. This means that new situations cannot be handled in absence of the user. CIP addressed exactly that problem by combining an anomaly detection with an active learning scheme, where the user is queried to incrementally add knowledge to the system. In other words, the user is asked to teach how to recover from the detected abnormal situation. As this can happen at any time, the user can seamlessly start with a basic task demonstration and add further knowledge in the future. In SBP, the user might not be aware that a transition between solutions is not feasible during execution. In UIP, the user might introduce a decision state at a time the robot cannot measure it or is not equipped with the sensors to do so, for instance a camera. Consequently, the task execution of such programs fails. In our proposed method, this problem is avoided by the active learning scheme, where the system communicates whenever it encounters difficulties (anomaly detection) to the user and then the user teaches the robot how to recover from that.

VI. CONCLUSION AND FUTURE WORK

We proposed an active learning framework that allows non-experts to intuitively program conditional tasks without writing code, involving sensor readings of force, motion and grasp status. We demonstrated that task conditions can be effectively demonstrated by our interactive programming scheme, where the robot asks for user input when a new environmental situation arises. Introducing the parameter-free on-line anomaly detection during task execution reduces not only the user's workload, but also ensures the creation of a functioning task model. The incrementally generated task-representing graph eliminates perceptual aliasing and guarantees a successful transition between skills in the *decision states*. This allows to scale the complexity of a task over time without reducing the overall performance.

As future work, a user study is planned to reveal more insights about the intuitiveness of each of the methods and its applicability for intuitive task programming that involves environmental conditions or recovery behaviors.

REFERENCES

[1] G. Biggs and B. MacDonald, "A survey of robot programming systems," in *Australian conference on robotics and automation*, 2003, pp. 1–3.

[2] C. Loughlin, A. Albu-Schäffer, S. Haddadin, C. Ott, A. Stemmer, T. Wimböck, and G. Hirzinger, "The dlr lightweight robot: design and control concepts for robots in human environments," *Industrial Robot: an international journal*, 2007.

[3] G. Maeda, M. Ewerton, T. Osa, B. Busch, and J. Peters, "Active incremental learning of robot movement primitives," in *Conference on Robot Learning (CoRL)*, 2017, pp. 37–46.

[4] S. Chernova and M. Veloso, "Confidence-based policy learning from demonstration using gaussian mixture models," in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, 2007, pp. 1–8.

[5] L. Sauer, D. Henrich, and W. Martens, "Towards intuitive robot programming using finite state automata," in *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*. Springer, 2019, pp. 290–298.

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

[7] D. Kappler, P. Pastor, M. Kalakrishnan, M. Wüthrich, and S. Schaal, "Data-driven online decision making for autonomous manipulation." in *Robotics: Science and Systems*, 2015.

[8] M. N. Nicolescu and M. J. Mataric, "Natural methods for robot task learning: Instructive demonstrations, generalization and practice," in *International joint conference on Autonomous agents and multiagent systems*. ACM, 2003, pp. 241–248.

[9] A. Sena and M. Howard, "Quantifying teaching behavior in robot learning from demonstration," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 54–72, 2020.

[10] T. Eiband, M. Saveriano, and D. Lee, "Intuitive programming of conditional tasks by demonstration of multiple solutions," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4483–4490, 2019.

[11] R. Zöllner, M. Pardowitz, S. Knoop, and R. Dillmann, "Towards cognitive robots: Building hierarchical task representations of manipulations from human demonstration," in *International Conference on Robotics and Automation*. IEEE, 2005, pp. 1535–1540.

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

[13] FRANKA EMIKA GmbH, Munich, Germany., "FRANKA Desk," Acccessed: July 07, 2020. [Online]. Available: https://www.franka.de/capability

[14] S. Calinon and A. Billard, "Active teaching in robot programming by demonstration," in *International Symposium on Robot and Human Interactive Communication*. IEEE, 2007, pp. 702–707.

[15] D. Kulić, C. Ott, D. Lee, J. Ishikawa, and Y. Nakamura, "Incremental learning of full body motion primitives and their sequencing through human motion observation," *The International Journal of Robotics Research*, vol. 31, no. 3, pp. 330–345, 2012.

[16] P. Pastor, M. Kalakrishnan, S. Chitta, E. Theodorou, and S. Schaal, "Skill learning and task outcome prediction for manipulation," in *International Conference on Robotics and Automation*. IEEE, 2011, pp. 3828–3834.

[17] P. Pastor, L. Righetti, M. Kalakrishnan, and S. Schaal, "Online movement adaptation based on previous sensor experiences," in *International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 365–371.

[18] M. Hersch, F. Guenter, S. Calinon, and A. G. Billard, "Learning dynamical system modulation for constrained reaching tasks," in *International Conference on Humanoid Robots*. IEEE, 2006, pp. 444–449.

[19] A. Pervez and D. Lee, "Learning task-parameterized dynamic movement primitives using mixture of gmms," *Intelligent Service Robotics*, vol. 11, no. 1, pp. 61–78, 2018.

[20] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Trajectory formation for imitation with nonlinear dynamical systems," in *International Conference on Intelligent Robots and Systems*, vol. 2. IEEE, 2001, pp. 752–757.

[21] R. Caccavale, M. Saveriano, A. Finzi, and D. Lee, "Kinesthetic teaching and attentional supervision of structured tasks in human–robot interaction," *Autonomous Robots*, vol. 43, no. 6, pp. 1291–1307, 2019.