Autonomous Robots manuscript No.
(will be inserted by the editor)

Collaborative Programming of Robotic Task Decisions and Recovery Behaviors

Thomas Eiband · Christoph Willibald · Isabel Tannert · Bernhard Weber · Dongheui Lee

Received: 27 July 2021 / Accepted: 05 September 2022

Abstract Programming by Demonstration is reaching industrial applications, which allows non-experts to teach new tasks without manual code writing. However, a certain level of complexity, such as online decision making or the definition of recovery behaviors, still requires experts that use conventional programming methods. Even though, experts cannot foresee all possible faults in a robotic application. To encounter this, we present a framework where user and robot collaboratively program a task that involves online decision making and recovery behaviors. Hereby, a task-graph is created that represents a production task and possible alternative behaviors. Nodes represent start, end or decision states and links define actions for execution. This graph can be incrementally extended by autonomous anomaly detection, which requests the user to add knowledge for a specific recovery action. Besides our proposed approach, we introduce two alternative approaches that manage recovery behavior programming and compare all approaches extensively in a user study involving 21 subjects. This study revealed the strength of our framework and analyzed how users act to add knowledge to the robot. Our findings proclaim to use a framework with a task-graph based knowledge representation and autonomous anomaly detection not only for initiating recovery actions but particularly to transfer those to a robot.

Thomas Eiband, E-mail: thomas.eiband@dlr.de,
Christoph Willibald, Bernhard Weber, Dongheui Lee
German Aerospace Center (DLR), Institute of Robotics and Mechatronics, Wessling, Germany

Isabel Tannert, Dongheui Lee
Technical University of Munich (TUM), Department of Electrical and Computer Engineering, Chair of Human-centered Assistive Robotics, Munich, Germany

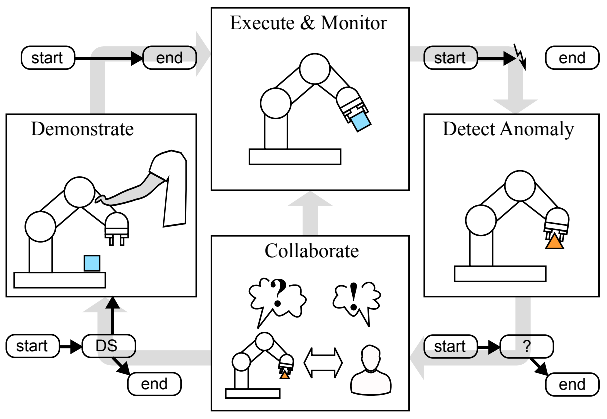


Fig. 1 Interactive Programming by Demonstration (PbD) framework for programming of task decisions and recovery behaviors, achieved by inserting decision states (DS) into a task-representing graph. In clock-wise order, a human provides an initial demonstration (left), the robot executes and monitors the current action (top), the robot detects a possible anomaly (right), human and robotic agent interact about how the new information shall be used (bottom) and either the robot executes again or the task structure is extended.

Keywords Learning from Demonstration · Programming by Demonstration · collaborative programming · interactive programming · anomaly detection · recovery behavior · task-graph · conditional task · force-based tasks · execution monitoring

1 Introduction

We are heading towards an age where robot programming is no longer subject to experts but requires shop floor workers and people in daily life situations to seamlessly program robots. It has been shown that Learning from Demonstration (LfD) is an intuitive technique to transfer task knowledge to a robot. More specifically, Programming by Demonstration (PbD) avoids manual code writing that is usually done by robotic experts [2].

Since we move from purely repetitive robot tasks used in manufacturing and assembly lines towards more adaptive, collaborative and intelligent robotic applications, there is a high demand to increase robustness and adaptability of the robotic behavior. An exemplary scenario is a workspace, which is shared between human and robot and where the human causes uncertainties or intentionally adapts object positions. One way to achieve robustness is a recovery behavior, where the robot has knowledge about how to resolve an erroneous state. Another way is to increase the adaptability to the environment with task decisions that are made based on the environmental state and that enable the robot to act in different ways. With that in mind, we are highly motivated to transfer such knowledge to a robot in an intuitive way, such that end users are capable of creating robot programs that include recovery behaviors and task decisions. To give examples for these scenarios, a robot could react to a failed grasp by a regrasping action or a robot could make a decision based on a specific object property, for instance, sort objects by their weight. Since conditions have to be monitored in an online fashion, these scenarios are also referred to as conditional tasks.

The goal of this work is to propose a new framework for programming conditional tasks, called *Collaborative Incremental Programming*, which is confronted with two other alternative approaches. To give a broader overview about the methodology of the compared frameworks, we structured it by two means, which are task representation and teaching interaction strategy. In the first part, we introduce two different task structures which represent the main task and its recovery behaviors or alternative actions. In the second part, we compare two different teaching interaction strategies that rely either on manual or automatic detection of erroneous states during execution. The comparison allows the analysis of intuitiveness and teaching efficiency for the end-user. This was achieved by conducting a study involving 21 users, which is evaluated by teaching and execution metrics as well as by user ratings.

The main contribution of this work is twofold: first, we propose a PbD task-graph learning framework that allows intuitive transfer of task knowledge including task decisions and recovery behaviors using a bidirectional communication channel between human and robot (see concept in Fig. 1). Second, we provide valuable insights of how users employ and understand PbD using different task representation structures and different interaction methods within a user study. In comparison to works that focus on the teacher's efficiency (e.g. [23]), we introduce a new programming framework and analyze how end-users collaborate with the robot as an autonomous agent via textual dialogs to achieve their task goals.

In our experimental evaluation, we show the advantage of our newly task-graph based method over an unstructured task representation in terms of robust and semantically consistent action transitioning. Further, we evaluate our anomaly detection method that relies on the robot's perception capabilities in comparison to a user-triggered anomaly detection. Our findings suggest that end-users have a biased impression about the robot's sensing capabilities, even though they were informed about them before usage.

This work gives a more detailed overview of our preliminary study on collaborative programming [26] regarding task representations, evaluates the framework in different applications and adds a user study in order to reveal how people interact with different frameworks.

2 Related Work

It has been early shown that PbD is a reasonable method for robot programming systems [14], which is also employed in our proposed framework. According to the problem to be solved, PbD can allow non-expert users to intuitively set up a new robotic tasks in comparison to manual programming. More recently, it has been shown that PbD can be successfully combined with other task learning methods such as human feedback and transfer learning of similar tasks [13]. After PbD has been established in the state of the art, researchers came up with structured representations of tasks, for example in the form of task-graphs [25,22,16,17,1]. In the presence of humans, who might cause uncertainties in the workspace or given a rather complex task, the robot requires some robustness to reach the task goal. In the work of Caccavale et al. [1], this has been achieved by the structured task representation on a visual perception level, where only branches of the task-graph are executed that are feasible for the robot at the given environmental state. A collaborative robot programming framework has been presented in [12] which uses augmented reality projections and a touch-enabled table to intuitively parameterize an existing robot program. The program itself allows preprogrammed branching or cycling operations. We enable the user to program branching operations by learning such behaviors from scratch without predefined skills, objects and environmental conditions. As the environment is not always fully observable and properties such as forces cannot be observed beforehand, we present a reactive task-graph-based framework that encounters unknown states with recovery behaviors that can be defined by the user.

2.1 Task Decisions

Several works have shown a sequential programming paradigm, where the robot executes a sequence of actions or skills in order to achieve the task [6,18,24]. However, a fixed sequence of actions is not able to solve conditional tasks, since it does not include replanning or decision making on the task level. Therefore, we introduced in a prior work [5] how intuitively task decisions can be programmed by demonstration, termed as *Sequential Batch Programming (SBP)*. Compared to this work, we substantially changed the way of task encoding and user interaction to allow a robust execution that is able to cope with unseen task faults. Although replanning of a robotic task during execution is possible, it requires a goal definition and world representation for the planner to work. We instead use the demonstrations itself to transfer the decision making strategy to the robot, which directly learns the required actions from the user. With that strategy, we enable both the definition of task decisions and recovery behaviors within the same framework.

2.2 Fault detection and recovery

In the context of fault detection and recovery, a variety of methods and applications have been presented. First, considering only fault detection, a method based on force data to train a Support Vector Machine has been applied to detect failures during assembly of a shield onto a counterpart [21]. In Pastor et al. [20], task outcome of failure or success is predicted by a statistical model of previous sensor signals. A Hidden Markov model (HMM) approach has been used to classify abnormalities in the force domain of an assembly task [3]. Also based on HMM, a multi-modal abnormality detection has been presented in [19] that monitors forces, vision and sound during execution. Khalastchi et al. presented a data-driven anomaly detection approach based on dimensionality reduction of sensor data, pattern recognition and a threshold on the Mahalanobis distance [10] and extensively evaluates this approach later on [9]. These approaches have in common that they are able to detect abnormal states or faults but are not designed to recover from them automatically. [4] proposed the derivation of recovery behaviors from geometric models of the task at hand. We do not require a geometric, predefined task model within our learning framework but extract the recovery actions directly from the user's demonstrations. [17] presented the construction of a finite state machine from a number of human demonstrations. Possible recovery behaviors were only considered, if the human pressed a button during execution. In contrast, our presented system decides autonomously when a demonstration is required via anomaly detection. Further, they provide the pose of all task relevant objects to the robot, which is hard to realize in practical applications. In [11], low confidence task regions based on a probabilistic model were exploited to improve the robot's spatial generalization capabilities for unseen object locations. Although no anomaly detection is performed online, the robot's knowledge about known motions is analyzed offline in order to request additional user demonstrations that could prevent future execution errors. In both [17] and [11], the force domain is not considered in the task definition process. Since we put a high emphasis on anomaly detection including the force/torque domain, we enable our framework to react to environmental properties that cannot be observed visually.

2.3 Sequential Batch Programming (SBP)

SBP is based on the framework presented in [5], 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 (see Fig. 2a). 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 the current sensor values and all alternative solution states. This error metric is computed by the Mahalanobis distance, that incorporates a confidence bound around each solution. The confidence bound is obtained by encoding multiple demonstrations per solution in a Gaussian Mixture Model (GMM).

2.4 User-triggered Incremental Programming (UIP)

UIP is inspired by the framework presented in [22] that suggests a robot state automaton which is able to observe environmental conditions and to branch into different states during execution. We adapted this approach in a way to only create graph-nodes where a decision state is required in order to obtain a task-graph (see Fig. 2b). Ordinary robot states within a trajectory are not represented as graph nodes, which allows to visually represent the task-graph with only the significant decision states. Similar to the approach we present, a task-representing graph is incrementally constructed in a combined teaching- and execution phase. The difference is, that with UIP, the teacher has to detect anomalies during execution of the task and needs to decide if and when a new demonstration is needed. A decision state can be inserted by manually triggering a button or controlling a GUI. In contrast, we tackle this problem by autonomous anomaly detection to remove this burden from the user.

3 Background: Programming of Recovery Behaviors by Demonstration

3.1 Requirements

We argue that a task decision and recovery behavior programming framework requires the following properties:

(i) an anomaly detection mechanism (Sec. 4.2),
(ii) an extendable knowledge representation allowing to learn from the user and environment (Sec. 4.3),
(iii) adaptability and refinement of robotic actions to increase robustness (Sec. 4.3, and
(iv) an adaptive system to react during task execution (Sec. 4.4).

According to that, we developed the approach of *Collaborative Incremental programming* (CIP) and compare it with two other approaches we have developed in this domain, namely *Sequential Batch Programming* (SBP) and *User-triggered Incremental Programming* (UIP).

3.2 Task Representations

We evaluate different task representations in this work that allow reactive behaviors that are required for fault recovery or conditional tasks. We clarify that fault recovery and conditional tasks are closely related, because they require (a) monitoring of the execution, (b) branching from the nominal execution flow, and (c) multiple actions for each decision and recovery behavior. In the following, two fundamental task representations are considered.

3.2.1 Solution Pool

This task representation has been introduced in our previous work [5] and represents a storage of multiple actions, so called solutions (Fig. 2a). In the solution pool, no branching states are specified, which enables transitioning between solutions at any time during execution.

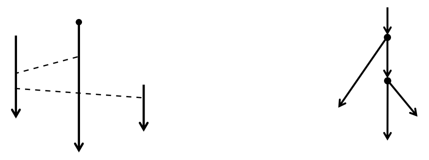

(a) Solution pool with nominal solution (middle arrow) and possible transitions (dashed lines) to alternative solutions.
(b) Task-Graph where links represent actions and nodes represent decision states.

Fig. 2 A task representation that incorporates recovery behaviors can be defined as solution pool, where multiple solution actions exist in parallel (a) or as task-graph, which arranges the actions as links and decision states as nodes (b).

3.2.2 Task-Graph

In this work, we make use of a structured task-graph (Fig. 2b), that employs specified decision states, which are the graph's nodes. The links represent the robotic actions that either lead to the next decision state or to a designated termination of the task. Later in this document, we explain how this representation can be generated incrementally in an interactive scheme involving user and robot.

3.3 Fault State Detection Mechanisms

In the presence of possible task faults, the end user wants the robot to handle such situations autonomously. In reality, it might not be always clear to the robot what is exactly a fault or erroneous state. However, a user might have capabilities that the robot has not in order to identify such states. Therefore, we consider both manual and autonomous detection mechanisms in this work.

3.3.1 Manual Fault State Detection

It has been shown that users are able to manually identify states where the robot shall make a decision about its next action in a specific environmental state [17,22]. This can be achieved by letting the user observe the task execution and by providing manual user feedback, e.g. via a button or GUI.

3.3.2 Autonomous Anomaly Detection

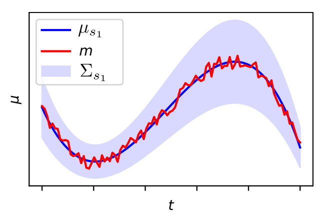
This detection scheme removes the burden from the user to observe the task execution and react accordingly. It enables detection of abnormal states in absence of the user and of newly occurred situations that could not be foreseen at programming time. In contrast to the identification of low confidence task regions to improve the robot's spatial generalization capabilities [11], we focus on the identification of anomalies that can occur in the position and force domain. We introduced our anomaly detection scheme in our previous works [5] and [26], which is based on a probabilistic action encoding and a statistical outlier detection using the Mahalanobis distance. The next section introduces all parts of our methodology in depth.

4 Collaborative Incremental Programming

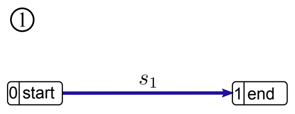
Our proposed approach of *Collaborative Incremental Programming* (CIP) combines the task-graph programming with an autonomous fault detection scheme that requests new user demonstrations in unknown regions of the input space. This enables the robot to decide ad-hoc when new information is required in order to extend the task-graph with decision states and possible recovery behaviors.

4.1 Probabilistic Action Encoding

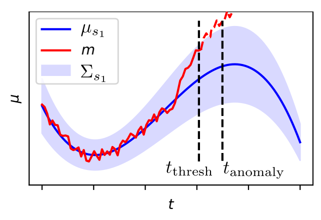
We request the user to only demonstrate a new behavior once, in order to add a new action. Since the dynamics of the kinesthetic demonstration differ slightly from the robot execution, we record also a robotic repetition of the given demo. Variations between user and robot performance are introduced by small uncertainties in the environment that are possibly introduced by the user, who sets the objects back to their original positions. This shall enable the anomaly detection to handle task-specific uncertainties that are possibly caused by uncertain object locations. Additionally, the anomaly detection shall be robust to system-specific uncertainties as they are caused by the robot controller due to limited tracking performance and variations in dynamics, depending on the stiffness parameters of the impedance controller. The obtained trajectory samples of the task are used to encode this action and determine the regions of variance around the nominal trajectory. An example of these variance regions can be seen in Fig. 3a. Hereby, low variance regions lead to a more sensitive anomaly detection. In parts with more variability, higher deviations are accepted during the execution, which increases the overall robustness. We make use of the robot's own proprioceptive sensing capabilities, where we use a force-torque sensor at the end-effector, the Cartesian pose and the signals from the gripper, which are the distance between the gripper fingers and the status informing if an object is grasped



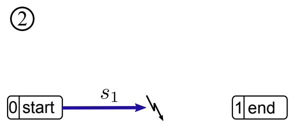
(a) Normal execution.



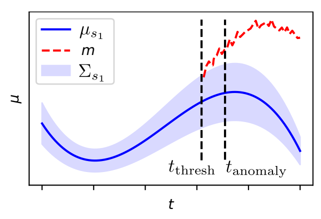
(b) Initial graph after first action encoding.



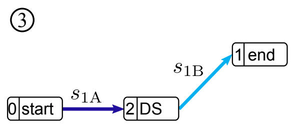
(c) Anomaly detection.



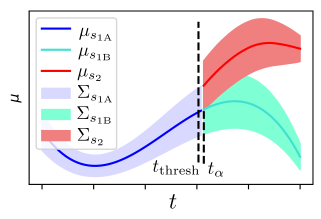
(d) Robot stops execution at abnormal state.



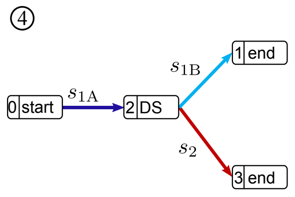
(e) Demonstration of new action.



(f) Insertion of a decision state (DS) by splitting the current action.



(g) Splitting up action s_1 at t_α.



(h) Insertion of an alternative action starting at the decision state.

Fig. 3 States while creating a task-graph by monitoring the execution and by reacting to anomalies.

or not, evaluated by the grasping force. An external vision system is not required in our approach, which performs well in partially structured production environments and hence is independent from object visibility or lighting conditions.

A data sample at time t is given as

$$\boldsymbol{x}_t = [\boldsymbol{p}, \boldsymbol{o}, \boldsymbol{f}, \boldsymbol{\tau}, g, h]^T \in \mathbb{R}^{15},$$

consisting of the end-effector's Cartesian position $\boldsymbol{p} = [x, y, z]$ and 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 $\boldsymbol{\tau} = [\tau_x, \tau_y, \tau_z]$, as well as the gripper finger distance g and grasp status $h \in \{-1, 0, 1\}$. The grasp status is defined as follows: $h = -1$ for no object in gripper, $h = 0$ for gripper closing or opening, and $h = 1$ for object in gripper. We choose these state variables since they were offered from the gripper hardware interface. The data is recorded at a frequency of 1 kH and is downsampled to 50 Hz to reduce the computational effort in learning. The recorded data from user demonstration $\boldsymbol{X}_{\text{Udem}} = [\boldsymbol{x}_{\text{U},1}, ..., \boldsymbol{x}_{\text{U},N_{\text{U}}}] \in \mathbb{R}^{15 \times N_{\text{U}}}$ and robot repetition $\boldsymbol{X}_{\text{Rrep}} = [\boldsymbol{x}_{\text{R},1}, ..., \boldsymbol{x}_{\text{R},N_{\text{R}}}] \in \mathbb{R}^{15 \times N_{\text{R}}}$ with respective sample length N_{U} and N_{R} is collected for each new demonstration.

Similar to [5], we first apply dynamic time warping (DTW) to align the two sensor sequences on a common time axis and equalize their length N. In a preceding step, the data is standardized dimension-wise with the z-transformation by subtracting the mean and dividing by the standard deviation. This assures that each dimension contributes equally to the dynamic time warping error. After warping the data, the standardization is undone by applying the inverse z-transformation dimension-wise.

In the next step, Expectation Maximization (EM) is used to learn a multivariate, time-based Gaussian Mixture model (GMM) for the input matrix

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

for an action s and a time vector $\boldsymbol{n} = [1, ..., N]$. The variables $\boldsymbol{X}_{\text{U}}$ and $\boldsymbol{X}_{\text{R}}$ refer to demonstrated and repeated trajectories respectively, where possible scenarios are explained in detail in Sec. 4.3. The model complexity is chosen such that the number of model components k is proportional to the temporal length N of the demonstrated time series data. In the experiments, we chose to add one model component per second of the time series, which has shown to be a reasonable trade off between model accuracy and smoothing of demonstrated motions. The EM algorithm is then initialized using k-means clustering with a number of k clusters. From here, we obtain a model $\mathcal{M} = GMM(\boldsymbol{G}_s)$ that can be used to reproduce a trajectory. Gaussian Mixture Regression (GMR) is applied to reproduce a generalized trajectory

$$\boldsymbol{Y}_s = [\boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_N] \in \mathbb{R}^{15 \times N}$$

with an associated sequence of covariance matrices

$$\boldsymbol{Z}_s = [\boldsymbol{\Sigma}_1, ..., \boldsymbol{\Sigma}_N] \in \mathbb{R}^{15 \times 15 \times N}.$$

These results allow the execution of the mean trajectory with a controller and to monitor the execution within a confidence area that is derived from the covariance matrices.

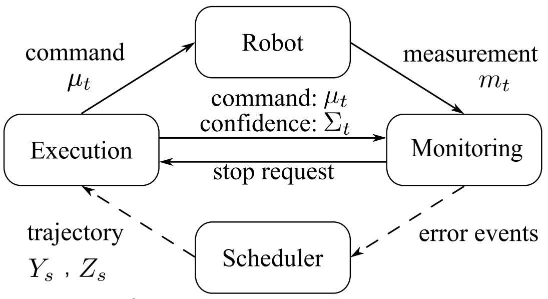


Fig. 4 System components for realtime execution and monitoring. Solid connections are realtime-capable up to 1 kHz, dashed connections are slow asynchronous connections.

4.2 On-Line Anomaly Detection

The system design for on-line anomaly detection is shown in Fig. 4. The main goal is to monitor the execution and detect new situations that are not known to the system. Hereby, sensor modalities are introduced to distinguish also the source of error. These modalities are

1. the robot pose $(\boldsymbol{p}, \boldsymbol{o})$,
2. the wrench $(\boldsymbol{f}, \boldsymbol{\tau})$, and
3. the gripper opening g and grasp status h.

For each of these modalities, the system constantly compares the commanded and measured values to detect abnormal states. Therefore, not only new situations can be detected but also a possible error source can be assigned, e.g. an abnormal state resulting from external forces. In each time step t of the execution, the deviation between the measurement $\boldsymbol{m}_{i,t}$ and commanded state $\boldsymbol{\mu}_{i,t}$ of a modality i is quantified using the Mahalanobis distance

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

By defining a custom anomaly threshold ϵ_i for each modality of an action s, this metric leads to a higher error sensitivity in time steps where the execution needs to be precise, indicated by small values of the reduced covariance matrix $\boldsymbol{\Sigma}_{i,t}$. During the execution, all modalities are monitored in parallel. If any $D_{\text{M}(i,t)}$ exceeds its action and modality specific anomaly threshold ϵ_i for e consecutive time steps, an anomaly is detected.

Our approach does not rely on manual error threshold tuning but is automatically parameterized from the training data. We compute an anomaly threshold ϵ_i for each modality of an action, based on the recorded trials of the user demonstration U and robot repetition R. After encoding a new demonstration in a GMM, we determine the highest occurring Mahalanobis distance for deviations between the samples $\boldsymbol{x}_{d,i,t}$ of each demonstration $d \in \{\text{U}, \text{R}\}$ belonging to one action and the associated mean $\boldsymbol{\mu}_{i,t}$ by

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

Then, the maximum distance over all trials is extracted with

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

and used as modality specific error threshold.

4.3 Collaborative and Incremental Graph Construction

We use a task-graph to structure the available robotic actions and possible decision states on an abstract level (such as shown in Fig. 3h). This graph is incrementally built by gaining task knowledge from user demonstrations. The graph's nodes represent system states that can be of type *start*, *end*, and *decision state (DS)* that is explained later on.

In order to construct a new task, a user triggers a demonstration phase and provides an initial task demonstration. A robotic action is extracted from this demonstration as explained in Sec. 4.1 (Fig. 3a). Next, a start and end state is added to the beginning and end of this action. The result can be seen in Fig 3b, which allows an execution of that simple task.

If an anomaly is detected during execution, as explained in Sec. 4.2, the robot stops at the unseen state (Fig. 3c and Fig. 3d). The system now queries the user to choose from the following options. The detected situation shall be handled by a new action in future executions (*Graph Extension*), or must be incorporated as refinement for the current action (*Action Refinement*). These two options are explained subsequently.

Graph Extension: If the user selects to add a new action that should resolve the current situation, the robot switches to a demonstration phase and waits for the user input. The robot configuration is still at the abnormal state and can now be changed by the user via kinesthetic teaching. We assume that an anomaly has been detected beforehand at timestep t_{anomaly}. In the following, a user demonstration $\boldsymbol{X}_{\mathrm{Udem}}$ is recorded. This data is appended to the time-series $\boldsymbol{M}$ that is recorded during the interval $[t_\alpha; t_{\mathrm{anomaly}}]$, resulting in $\tilde{\boldsymbol{X}}_{\mathrm{Udem}} = [\boldsymbol{M}, \boldsymbol{X}_{\mathrm{Udem}}]$. After finishing the demonstration, the user is requested by the system to restore the environment to the state before the demonstration, which means that manipulated object locations are set back to the beginning. Now, the robot moves to the configuration at time step t_α and repeats the extended user demonstration $\tilde{\boldsymbol{X}}_{\mathrm{Udem}}$. The two time-series from user and robot are then probabilistically encoded and saved as action s_2.

Finally, a new decision state is inserted into the graph, splitting up action s_1 into two actions before and after the anomaly, depicted $s_{1\mathrm{A}}$ and $s_{1\mathrm{B}}$ respectively (see Fig. 3g and Fig. 3h). The actions $s_{1\mathrm{B}}$ and s_2 are then appended to the newly inserted decision state. In detail, action s_1 is split at time step

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

where t_{thresh} is the time step in which the error metric $D_{\mathrm{M}(i,t)}$ first exceeds the anomaly threshold ϵ_i. The parameter e is the number of consecutive time steps for which $D_{\mathrm{M}(i,t)} > \epsilon_i$ until an anomaly is triggered. The scaling factor α $(0 < \alpha < 1)$ places the decision state in between time step t_{thresh} and t_{anomaly}.

An early and smooth transition from action $s_{1\mathrm{A}}$ to its successor without following a possibly erroneous strategy too long, requires a minimal α. This means that the decision state would be placed close to the timestep t_{thresh}. However, making a robust decision requires a long enough sequence of unambiguous sensor readings that can be assigned to a specific action, pushing the decision state towards t_{anomaly} and thereby $\alpha \to 1$. Furthermore, the decision for the subsequent action must be made before t_{anomaly} is reached during execution of action $s_{1\mathrm{A}}$, otherwise the anomaly detection would wrongly identify a new situation for the scenario handled by s_2 (see Fig. 3h). Preliminary experiments have shown that setting the number of error samples $e = 30$ (corresponding to $30/50\,\mathrm{Hz} = 0.6\,\mathrm{s}$) and the scaling factor $\alpha = 1/3$ is a good compromise between robustness and delay in decision-making.

Action refinement: In case the user wants to refine the action s, during which the anomaly was detected, its encoded trajectory $\boldsymbol{Y}_s$ with associated sequence of covariance matrices $\boldsymbol{Z}_s$ is adjusted by new data. Hereby, an existing action becomes capable of handling more diverse conditions such that the robot learns which features are important to observe and which regions of the state space do not require a tight monitoring and error handling. For instance, a sorting task for geometrically different objects, that ignores the object weight can be achieved by refining the actions that handle the different geometries. In that case, the refinement leads to actions, where the monitoring becomes invariant to the object weights and therefore avoids false-positive force anomaly detection in future task executions. Such an example is later on evaluated in the experiments section. A trial of the new setup is either acquired by a user demonstration in a *user-refine* mode or by the robot in *auto-refine* mode.

In *user-refine* mode, a manual demonstration offers the possibility to adjust the full trajectory of the correction, which directly starts at the anomaly configuration. In comparison, the *auto-refine* mode lets the robot autonomously continue the execution after the anomaly has been detected until the end of the action. Since we know already that a new situation shall be incorporated into the action encoding, the anomaly detection is disabled for the remainder of the execution. For both possible modes, the recorded time-series $\boldsymbol{X}_{\mathrm{Rref}}$ is appended to the time-series $\boldsymbol{M}$ of the action before the anomaly, resulting in a stacked matrix $\tilde{\boldsymbol{X}}_{\mathrm{Rref}} = [\boldsymbol{M}, \boldsymbol{X}_{\mathrm{Rref}}]$. This data is used together with the initial user demonstration $\boldsymbol{X}_{\mathrm{Udem}}$ and robot repetition $\boldsymbol{X}_{\mathrm{Rrep}}$ of that action for a new probabilistic encoding, as described in Sec. 4.1. Finally, the task-graph is updated with the new action model.

4.4 Task Execution

Our main goal is the efficient combination of teaching and execution phases, that switch ad-hoc according to changes in environmental conditions. After an initial task demonstration, an execution phase can immediately follow to start the production. It is seamlessly possible to add knowledge at any time to the task-graph. Either, the user can intentionally add knowledge for known situations from the beginning, or the system just comes back to the user at any time, for instance after unforeseen faults occurred during production. The task-graph enables the robot to reproduce any demonstrated task, but furthermore, allows to adapt to environmental states by exploiting known decision states. This allows a fundamental extension to a simple sequential task execution, which is namely the selection of the appropriate action based on the current sensor readings. Conditional tasks allow, for instance, sorting by object properties, or selection of recovery behaviors at erroneous states.

The task-graph structures the available actions on a high level, while the actions themselves are encoded probabilistically on a low level, enabling their realtime monitoring. Decision states are automatically inserted at critical state transitions of the task, which simplifies the decision process for a specific state, but also eliminates perceptual aliasing and thereby the risk of deciding for a wrong action. Since decision states are known after the first anomaly occurred, the system can evaluate the measurements in an early state and avoid unnecessary robot movements.

Our approach identifies the sensor modality that contributed most to the anomaly, where only relevant sensor values are considered to select the subsequent action in a decision state. In the following, an example is used to explain the action selection in a decision state, referred to Fig. 3h. The robot starts with the first action $s_{1\mathrm{A}}$. If no anomaly is detected during the execution, the robot reaches the first decision state (DS), in which the subsequent action $\hat{s}$ is determined by

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

$\boldsymbol{m}_{\mathrm{DS}}$ is the measured state of of a modality in the decision state and $\boldsymbol{\mu}_{s,0}$ is the sample of the same modality at the first time step of an encoded action. In our example, the action $\hat{s}$ that is executed next is selected from $\{s_{1\mathrm{B}}, s_2\}$, which are all actions that are attached to the decision state. In contrast to the anomaly detection, we use the Euclidean distance metric here, because the Mahalanobis distance favors actions with high uncertainty, expressed by large values in the covariance matrix that lead to very small errors in the first time step. With our proposed scheme, the robot always chooses an action that minimizes the error to the current environmental state and keeps on monitoring that action to detect possible future anomalies.

5 Experiments

Our experiment shows a scenario where a user transfers a sorting task to a robot by adding knowledge incrementally[1]. Hereby, the system queries only three demonstrations from the user by interacting via the GUI. If the robot detects an anomaly during task execution, the user can either demonstrate a new action that solves this unique situation or refine the current action by incorporating the new conditions into the expected outcome of that action. With this experiment we want to demonstrate both the action refinement and the task-graph extension capabilities of our approach allowing the robot to ignore irrelevant features and to learn relevant features of a task.

5.1 Experimental Setup

As seen in Fig. 5, a DLR LWR IV 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

[1] The accompanying multimedia material contains a video of this experiment (video 1).

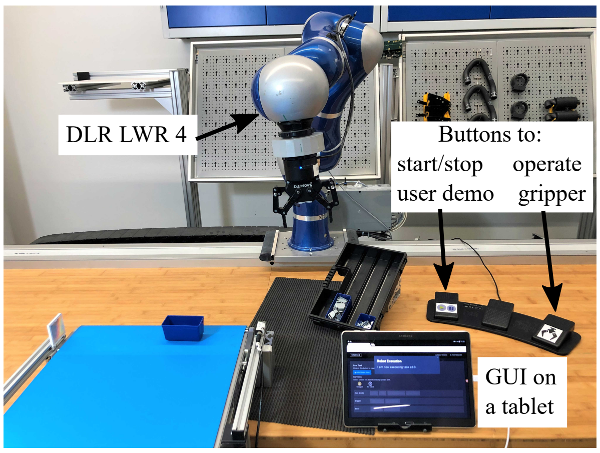


Fig. 5 Experimental setup where a conveyor belt (blue in the bottom left) delivers new parts to a pick location, from where they can be sorted.

acting on the end-effector. The robot is impedance controlled with a control frequency of 1 kHz and parameterized with constant stiffness- and damping coefficients $k_{\text{trans}} = 1200$ N/m, $k_{\text{rot}} = 100$ Nm/rad and $d_{\text{m}} = 0.3$ Ns/m respectively. Pedals and a tablet displaying a GUI allow the user to interact with the GUI while guiding the robot at the same time. The pedals are used to open or close the gripper and to start or stop the demonstration recording when using kinesthetic teaching in gravity compensation. The GUI guides the user through the teaching process and requests input from the user when the task definition requires it. A conveyor belt standing perpendicular to the table transports boxes with supplies for an assembly task to a determined place in the working space. These boxes have to be placed in a part storage on the table in front of the user.

5.2 Geometry-based Sorting Task

The goal of the task is to program the robot to distinguish the different boxes based only on their geometry in order to place the supplies at a specific spot in the part storage where the user expects them. Specifically, the weight of the boxes should not be considered when deciding for the final position of a box. Analogously, sorting of objects by their weight can be achieved with similar means, as realized in a previous work [5]. We assume that boxes of equal dimensions always contain the same kind of pieces but do not always contain the same number of pieces and therefore differ in weight.
First, the user provides an initial demonstration, where the robot picks up a box with supplies from the start position on the conveyor belt and places it in its designated spot in the part storage (Fig. 6d). After the user demonstration, the box is again placed in the start position on the conveyor belt so that the robot can repeat the demonstrated sequence. As seen in Fig. 6a-c, these two demonstrations are used to learn a model of the action, which is then executed by the robot. During manipulation of a box with a different weight, the robot detects an anomaly caused by an unexpected force f_z exerted in z-direction (Fig. 6e). Since deviating weights of boxes are not considered important features of the task, the user decides to refine the current action in order to incorporate the new condition into the expected trajectory for that action. The refinement is shown in Fig. 6h and carried out completely by the robot, continuing the learned motion and placing the box in its designated spot. However, when the robot detects a different box geometry during gripping (Fig. 6i), the user can demonstrate a new action, placing the box in another spot of the part storage (Fig. 6l). This user demonstration is then again repeated by the robot and encoded into a probabilistic model of the new action, shown in Fig. 6j.

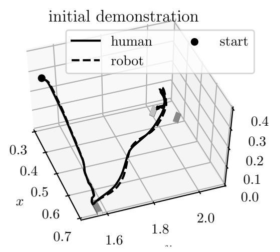
5.3 Results

We have shown that with our approach the robot can learn important features (box geometry), while considering deviations of other features (box weight) irrelevant for specific actions of a task. As seen in Fig. 6g, refining the learned model with an example of a lighter box adjusts the expected value of f_z and the variance σ_{f_z} as well as the force anomaly threshold ϵ_f for this action (acc. to Eq. (3) and (4)). Following Sec. 4.2, this leads to a less sensitive force anomaly detection in future executions of this action. This allows to manipulate boxes with a wide range of different weights without triggering a false positive anomaly detection. At the same time, the robot can still learn additional actions for new situations. As seen in Fig. 6i and 6l, a detected gripper finger distance anomaly when grasping a different box gives the user the opportunity to demonstrate a new action that places this box at another goal position. At the time step of grasping a box, a new decision state is inserted into the task-graph (Fig. 6j) in which the robot decides for the subsequently executed action based on the measured gripper finger distance (see Fig. 6k, Sec. 4.3 and Sec. 4.4).

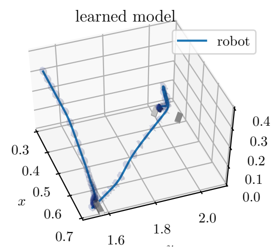
6 User Study for Approach Comparison

In order to evaluate the intuitiveness and user friendliness of CIP, a user study is conducted[2], in which it

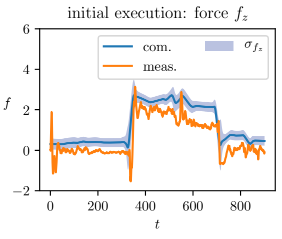
[2] The accompanying multimedia material contains a video of the experiments in the user study (video 2).



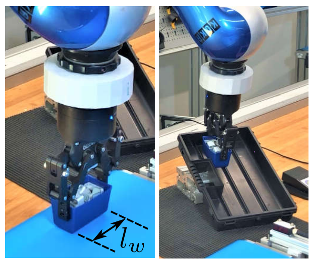
(a) Human demonstration and robot repetition for the initial task.



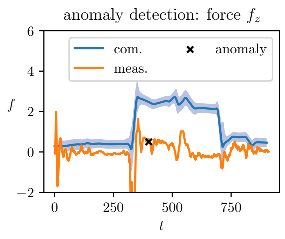
(b) Encoded action s_1 from the data shown in a).



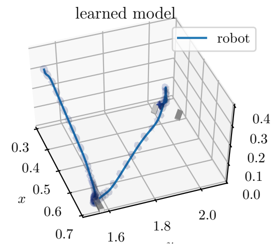
(c) Initial execution of s_1 with commanded (com.) and measured (meas.) time-series.



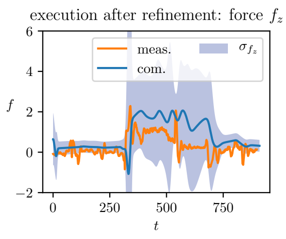
(d) Left: picking the wide box with edge length l_w, right: placing the wide box in the tray.



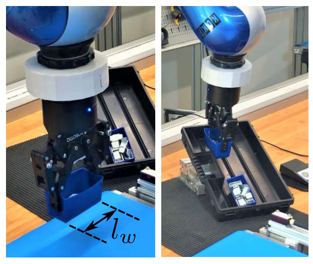
(e) Execution with anomaly detection in the force domain for action s_1 while picking a lighter box as before. The measured (meas.) time-series is used to update the model of this action.



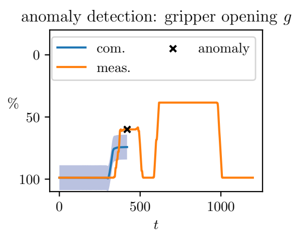
(f) Refined model of action s_1 considering the measured data shown in e).



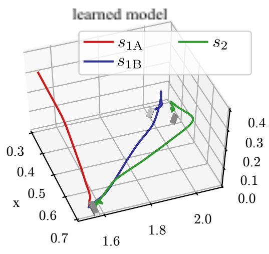
(g) Execution of s_1 with an increased error margin related to σ_{f_z}.



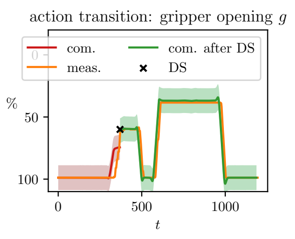
(h) Left: picking the lighter wide box with edge length l_w, right: placing the lighter wide box at the same location.



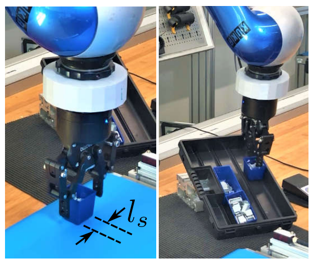
(i) Execution with anomaly detection in the gripper finger distance.



(j) Learned model after splitting action s_1 into s_{1A} and s_{1B} and added action s_2.



(k) Execution exploiting the decision state to switch from action s_{1A} to the new action s_2 that resolves the anomaly.



(l) Left: picking the small box with edge length l_s, right: placing the small box on a new location, specified by the user.

Fig. 6 Experimental results of the box sorting task with action refinement and graph extension.

is compared with two other frameworks, SBP and UIP that were introduced in the related work (Sec. 2). The task executions, generated with the different programming frameworks are finally compared by their performance in reaching the task goals.

6.1 Materials and Study Design

6.1.1 Sample

21 participants (19 male and 2 female) were recruited from the German Aerospace Center (Age = 25.24 ± 7.03 years, ranging from 21 - 56). All participants have a background in different technical fields, but not necessarily in robotics.

6.1.2 Setup

We use the same setup as described in the previous experiments section. For all robotic tasks in the user study, we use the same object, an aluminum block visible in Fig.7 (6.8 cm x 4 cm x 2 cm) in different setups.

6.1.3 Procedure

Participants are informed about the aim of the study and the procedure. In an introduction, the robot's sensing capabilities are explained, specifically highlighting that no vision-based monitoring of the environment is used. After up to five minutes to familiarize with handling the robot and operating the pedals, the experimental tasks are explained. Each participant watches a short instruction video explaining each method and then teaches both tasks for all three methods. The order of teaching each task with each method was permuted among all subjects using a Latin square design [7]. After programming with one method is completed, the NASA-TLX [8] and the Questionnaire for Measuring the Subjective Consequences of Intuitive Use (QUESI) [15] are filled out by the participants. With the end of the experiment, an overall evaluation of the methods takes place, where the participants rate intuitiveness and efficiency on a 7-point Likert-type scale followed by a semi-structured interview.

6.1.4 Data Analysis

Nominal scaled successful completions were analyzed by means of Cochran's Q test and McNemar post hoc tests in case of significant differences between methods. For questionnaire items, a repeated measures ANOVA was calculated. In case of violation of sphericity (Mauchly's sphericity test), Huynh-Feldt ($> .75$) or Greenhouse-Geisser ($< .75$) corrections were made. Post hoc tests with Bonferroni correction were performed to identify which methods differ significantly.

6.2 Compared Methods

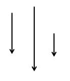
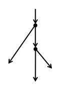
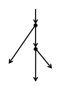
Table 1 provides an overview of the PbD approaches that are compared in the user study, which all use the same sensory input but no visual perception to make task decisions. The approaches were initially described in the related work sections (Sec. 2.3 and Sec. 2.4) and are briefly explained in the following.

Sequential Batch Programming (SBP) is based on the framework presented in [5], where the teaching and execution phases are separated. First, the teacher successively demonstrates all task solutions which the robot shall be able to handle, and stores these independently in a solution pool. If an anomaly occurs during task execution, the system switches to the state within an alternative solution that minimizes the error between current measurement 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 collaboration between user and robotic agent. Therefore, arbitrary switching states that do not guarantee a successful transition are avoided.

User-Triggered Incremental Programming (UIP) is inspired by the framework presented in [22], 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 execution of the task and needs to decide if and when a new skill demonstration is needed.

Table 1 Overview of Compared Frameworks

Methods \ Properties	Sequential Batch Progr. (SBP)	Collaborative Incremental Progr. (CIP) (ours)	User-trigger. Incremental Progr. (UIP)
task representation			
teaching-interaction	unidirectional	bidirectional	unidirectional
incrementally extendable	✘	✔	✔
online decision making	✔	✔	✔
autonomous anomaly detection for programming	✘	✔	✘

6.3 Hypothesis

In this study, we want to verify the following hypotheses:

- $\mathcal{H}$**1** (based on objective metrics): Using CIP with its collaborative programming concept and autonomous anomaly detection results in a significant increase in successful task completions,
  - compared to SBP (hypothesis $\mathcal{H}$**1.1**), and
  - compared to UIP (hypothesis $\mathcal{H}$**1.2**).
- $\mathcal{H}$**2** (based on subjective ratings): A significant increase in programming intuitiveness is achieved by CIP with its collaborative programming scheme,
  - compared to SBP, which uses a training phase to collect all demonstrations in the beginning (hypothesis $\mathcal{H}$**2.1**), and

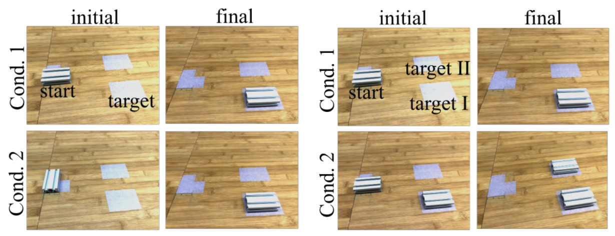


Fig. 7 Initial and final setups of task 1 (left) and task 2 (right) with each two different environmental conditions (Cond. 1 and Cond. 2).

  - compared to UIP, which requires the user to trigger the insertion of decision states manually (hypothesis $\boldsymbol{\mathcal{H}}\mathbf{2.2}$).
- $\boldsymbol{\mathcal{H}}\mathbf{3}$ (based on subjective ratings): A significant decrease in workload is achieved by CIP,
  - compared to SBP (hypothesis $\boldsymbol{\mathcal{H}}\mathbf{3.1}$), and
  - compared to UIP (hypothesis $\boldsymbol{\mathcal{H}}\mathbf{3.2}$).

6.4 Experimental Tasks

We designed two different tasks, namely task 1: *Reorientation* and task 2: *Contact-based Sorting*. Their initial and final setup is shown in Fig. 7. In task 1, the robot shall manipulate an object from an initial location 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. 8 and Fig. 9. In task 2, the robot shall 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. 12.

6.5 Results

Methods were evaluated using objective performance data and subjective user feedback in post-experimental questionnaires and the interview.

6.5.1 Objective Data

Successful Completions: A binary metric was used to determine if a learned task can be successfully executed in order to reach the task goal as described in the experimental task description. This allows to compute the success rate of executions for each method, as shown in Fig. 14. Cochran's Q test indicated significant differences between the conditions for task 1 ($p < .001$) as

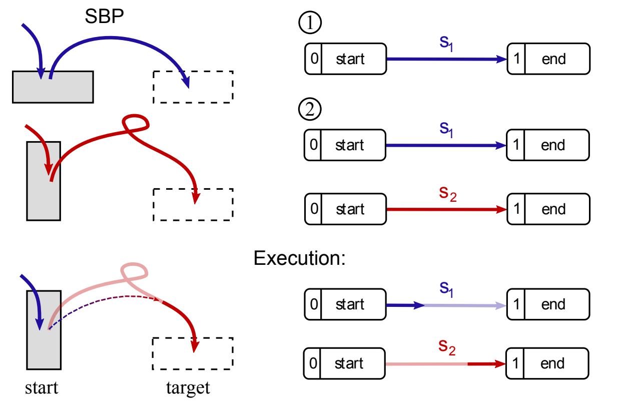


Fig. 8 Task 1: Reorientation, SBP: In step (1), the user demonstrates a pick and place action s_1. In step (2), the user extends the solution pool with a second action s_2, in which the object gets rotated by 90° before placing it in the target location. During execution of the nominal solution s_1, 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.

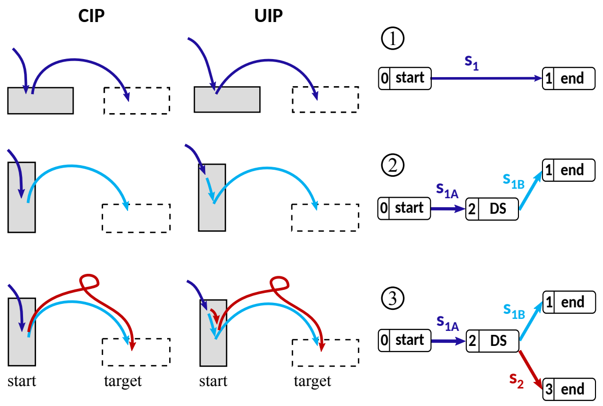
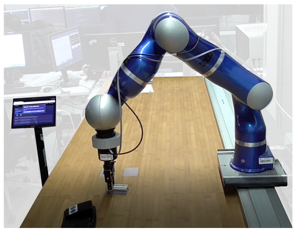


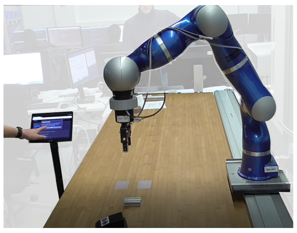
Fig. 9 Task 1: Reorientation, CIP and UIP: In step (1), the user demonstrates a pick and place action s_1. Step (2) shows the updated graph after first execution where an anomaly leads to inserting decision state (DS) and splitting s_1 into s_{1A} and s_{1B}. The DS is created by the anomaly detection algorithm in CIP and by the user manually in UIP. In step (3), the user adds a new action s_2 that accounts for the anomaly and properly rotates the object before placing it.

well as for task 2 ($p < .001$). McNemar post hoc tests revealed significant differences between SBP and CIP ($p < .05$) and CIP and UIP ($p < .001$) for task 1. For task 2, significant differences could be found for SBP versus UIP ($p < .001$) as well as CIP versus UIP ($p < .001$).

$\boldsymbol{\mathcal{H}}\mathbf{1.1}$ does not hold for task 1 (✘) but holds for task 2 (✓) such that there are significantly more successful task completions by using the collaborative programming scheme of CIP compared to the collection of demonstrations in a batch, used in SBP. This could



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



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

Fig. 10 Correct (a) and wrong (b) robot configuration to provide an alternative action 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.

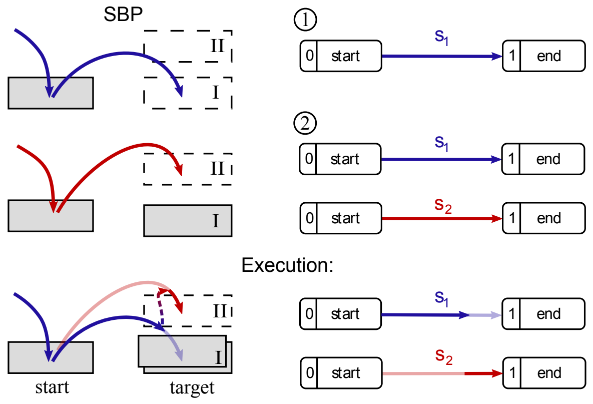


Fig. 11 Task 2: Contact-based Sorting, SBP: The user successively demonstrates two pick and place actions in step (1) and (2). In demonstration of action s_2, the object is placed in target location II, if target location I is occupied by another object. The bottom row shows the execution of the nominal 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.

be explained by the importance of right timing in task 1 (Reorientation), where it was critical for SBP to find the precise entry point in the alternative solution, which could lead to failed grasps and an unsuccessful task outcome. In task 2, this timing issue was less critical as the recovery behavior did not grasp the object again, but just executed an action with different trajectory while the gripper remained closed. $\mathcal{H}$**1.2** holds for both tasks (✓) with significantly more successful task completions by using the autonomous anomaly detection of CIP in favor of a manual anomaly detection in UIP. Due to this discrepancy in the success rates, we analyzed where exactly the decision states were inserted in these approaches.

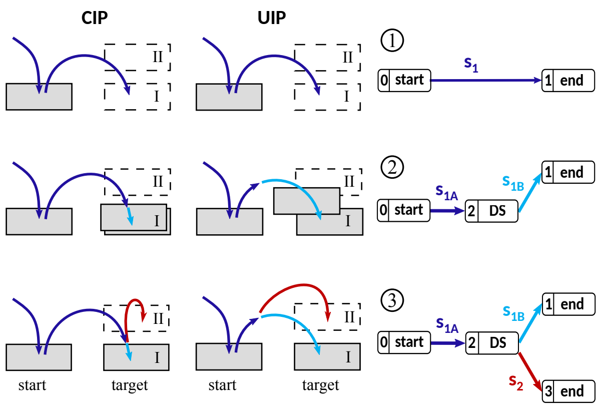


Fig. 12 Task 2: Contact-based Sorting, CIP and UIP: Step (1) shows the initial demonstration of a pick and place action s_1. Step (2) shows the updated graph after first execution where an anomaly leads to decision state (DS) insertion and splitting of s_1 into s_{1A} and s_{1B}. The DS is created by the anomaly detection algorithm in CIP and by the user manually in UIP. In step (3), the user added a new action s_2 that recovers from the anomaly.

Decision State Insertion: The timestep where the anomaly is detected defines where the decision state is inserted in the task-graph. This is critical for selecting the appropriate action from the task-graph during execution. This timestep reflects a specific position of the end effector. In both tasks, the position of the end effector at the decision state is the main constraint to allow force sensing or grasp status identification of an object. To analyze this further, we derive a ground truth for the position of a decision state for each task. Hereby, we store the end effector position of all decision states from successful task executions of both CIP and UIP. Next, we compute the mean over all stored positions. This serves as ground truth, which can be considered as a near optimal solution to solve the task. Finally, we compute the distance $d_{EE,C}$ between the end effector position of each decision state and the ground truth and show these values as green marks in Fig. 15. With CIP (left column), the automatically identified decision states lie close to the ground truth while with UIP, these were manually inserted and show larger errors. These errors lead to decision states, that are not physically grounded because the targeted sensor signal is not present in that state. Imagine that a user manually triggers a decision state that should decide about the weight of an object before the robot actually grasped it, which makes it impossible to sense such property.

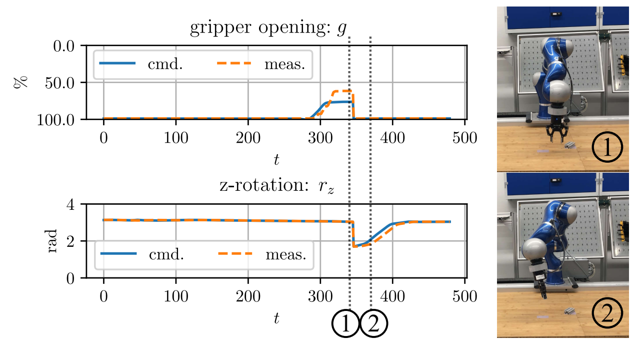
6.5.2 Subjective Data

QUESI ratings: As reported in Fig. 17, users rated the intuitive use of SBP best, followed by CIP, except for "perceived achievement of goals", where CIP reached

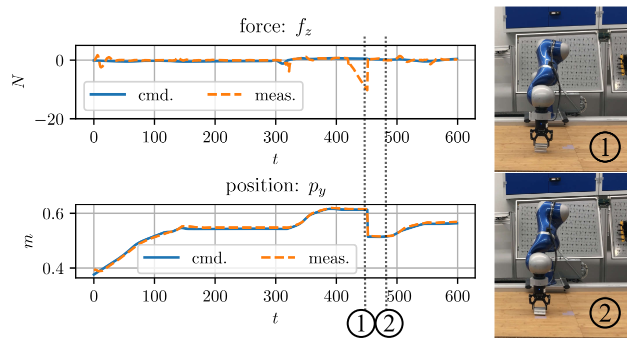
Task 1: Reorientation

Task 2: Contact-based Sorting

SBP

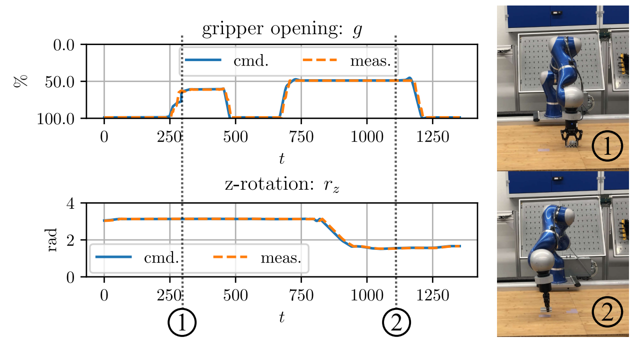


(a) At timestep (1), the robot detects an unexpected griper opening that triggers switching to an alternative action. The robot chooses a wrong entry timestep in the alternative action, thus skipping the reorientation part, which leads to an unsuccessful task execution (2).

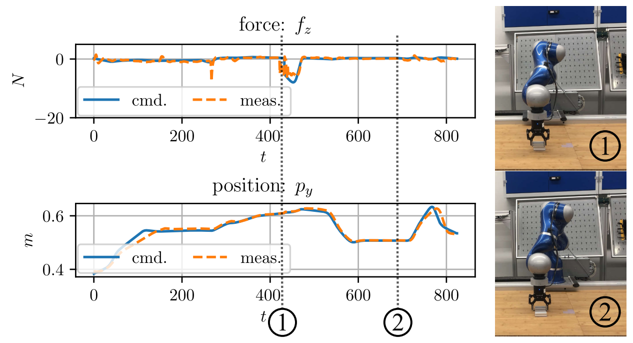


(b) When trying to place the object in the occupied target location I, the robot senses an unexpected force in z-direction (1) that triggers switching to an alternative action. The robot transitions to a correct entry timestep in the alternative action, adjusting the end effector's y-position (2) before successfully placing the object in target location II.

CIP

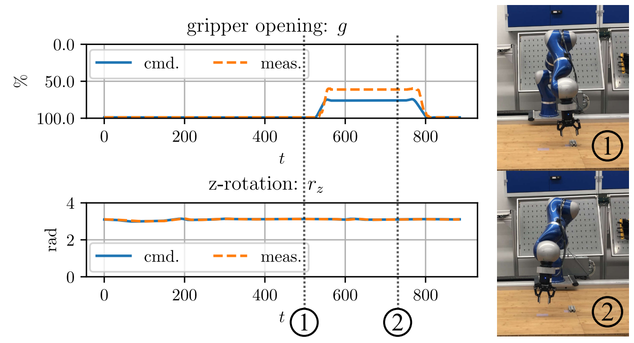


(c) When grasping the object, the robot decides for the subsequent action s_2 in the decision state (1) based on the measured gripper opening. Using action s_2, the robot rotates the object before successfully placing it in the goal location (2).

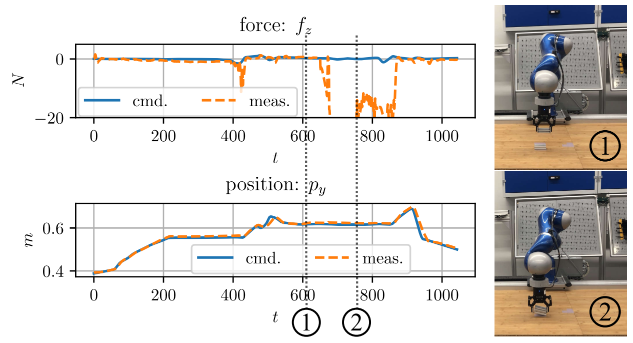


(d) When trying to place the object in the occupied target location I, the robot senses a contact force in z-direction in the decision state (1) and decides for the subsequent action s_2. Using action s_2, the robot adjusts the end effector's y-position (2) before successfully placing the object in target location II.

UIP



(e) In the decision state (1), the next action is chosen before grasping the object, at a time step in which the robot does not interact with the environment and thus cannot sense a difference between action s_{1B} and s_2. The robot selects the unsuited action s_{1B} for this situation, which leads to an unsuccessful task execution (2).



(f) In the decision state (1), the next action is chosen before the object in target location I can be detected by a contact force in z-direction. The robot selects the unsuited action s_{1B} for this situation and tries to place the object in the occupied target location (2), thus leading to an unsuccessful task execution.

Fig. 13 Exemplary executions from the user study experiments.

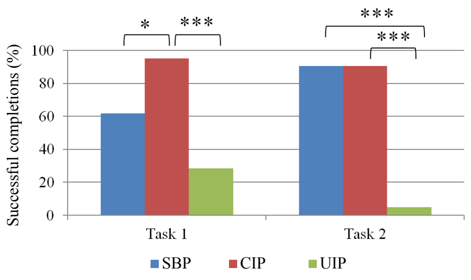


Fig. 14 Successful completions of the different tasks for all three methods in percent.*, p < .05; **, p < .01; ***, p < .001.

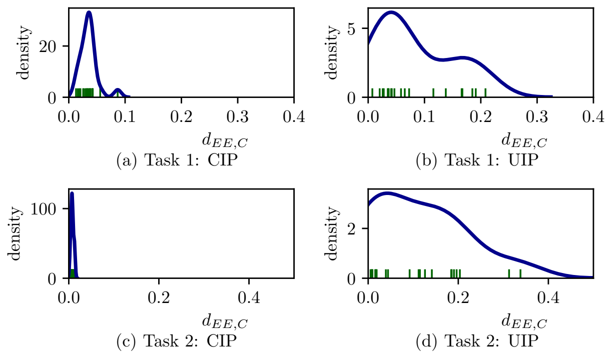


Fig. 15 Each plot shows the probability density (blue curve) for the computed distances between decision state position and ground truth. These distances are marked by the green samples on the x-axis. The left column displays the automatically detected decision states by CIP, while the right column shows the manually triggered decision states by UIP. Automatically detected states (left column) lie notably closer to the ground truth.

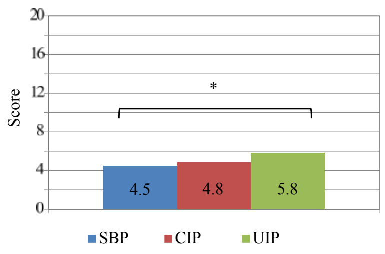


Fig. 16 NASA-TLX workload

the highest score. UIP was rated worst for all scales. A repeated measures ANOVA showed that statistically significant differences occurred for the subscales “Subjective Mental Workload” (F(1.37, 27.39) = 5.36; p < .05), “Perceived Effort of Learning” (F(1.38, 27.67) = 5.39; p < .05) and “Familiarity” (F(2, 40) = 4.09; p < .05). Post-hoc comparisons showed that SBP scored higher for those items than UIP (“Subjective Mental Workload”: p < .001; “Perceived Effort of Learning”: p < .05; “Familiarity”: p < .05) (see Fig. 17).

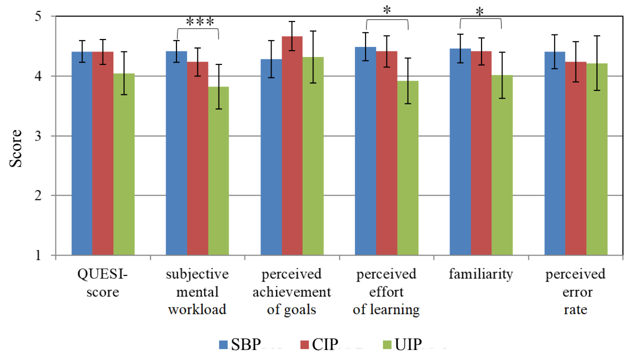


Fig. 17 Scores for QUESI (error bars indicate 95% confidence intervals).*, p < .05; **, p < .01; ***, p < .001.

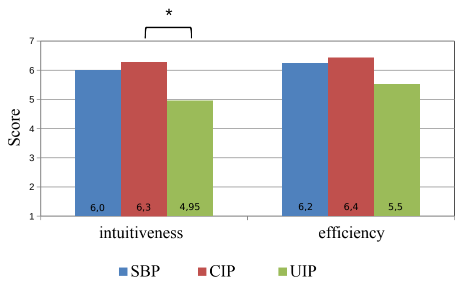


Fig. 18 Scores for overall evaluation. p < .05; **, p < .01; ***, p < .001.

Workload: NASA-TLX overall score (see Fig. 16) revealed a significant ANOVA main effect (F(2, 40) = 4.30; p < .05). With post-hoc comparisons we found a significant lower workload for SBP (M = 4.48; SD = 2.21) compared to UIP (M = 5.82; SD = 2.79; p < .05). No significant difference was evident comparing CIP (M = 4.81; SD = 2.29) to any other method.

$\mathcal{H}$3.1 suggests that the programming workload is reduced by CIP in comparison with SBP and ***$\mathcal{H}$3.2*** suggests the same effect for the comparison of CIP with UIP. Both hypotheses were rejected, instead we only see a significant difference between SBP and UIP. That SBP shows the smallest workload rating could be explained by a minimum of required human-robot interactions, where all knowledge is transferred sequentially in the teaching phase before the robot executes the task.

Overall Evaluation: The user ratings for the following two items are shown in Fig. 18. *Intuitiveness of the method (“The method was easy to use and intuitive”).* CIP (M = 6.29; SD = 1.35) and SBP (M = 6.00; SD = 1.10) were more intuitive than UIP (M = 4.95; SD = 1.69). This is supported by a significant ANOVA main effect (F(2, 40) = 4.89; p < .05), where CIP and UIP significantly differ (p < .05). Conventional level of significance for the difference between SBP and UIP was not reached (p = .053).

$\boldsymbol{\mathcal{H}}$**2.1** that suggests a higher intuitiveness of CIP compared to SBP in programming a task is supported by the overall QUESI ratings but without statistically significant effect (Fig. 17 very left). In contrast, $\boldsymbol{\mathcal{H}}$**2.1** holds for the comparison of CIP with UIP (✓) and shows a significantly higher intuitiveness in programming a task.

Efficiency of the method ("I could solve the given tasks efficiently with the method"). Subjects rated CIP (M = 6.43; SD = 0.98) as most efficient, followed by SBP (M = 6.24; SD = 1.09). UIP (M = 5.52; SD = .47) was slightly less efficient. However, this is not supported by a significant ANOVA effect.

6.6 Discussion

6.6.1 Objective Data

The results from the performance evaluation show, that only programs created with CIP reliably solved both experimental tasks.

Due to the different abilities of the user and the robot to perceive the environment (e.g. vision), UIP cannot guarantee that the robot will be able to measure abnormal values when the user identifies a new situation and demonstrates an alternative behavior. As seen in Fig. 10b, during the experiments, many subjects did not wait with a demonstration until the robot senses the transition condition for the second sub-task. When programming the Reorientation task, 13 participants demonstrated a new action before the robot closed the gripper to grasp the turned object. For the contact-based sorting task, even 16 subjects did not wait until the robot could detect an object in the target location. With CIP, however, a deviation in sensor values is a requirement for detecting new situations. Thereby a measurable difference between the programmed transition conditions for every action of a *decision state* can be guaranteed. This leads to a successful transition to the appropriate successor action when reproducing the situation, because the measured sensor values reflect a programmed condition for action transitioning.

With SBP, a transition between actions is triggered when an anomaly is detected during the task execution. The time step of an action with the closest sensor values to the anomaly state is chosen as an entry point to continue the task. Since all time steps of all actions are potential candidates for the entry point, the approach is prone to perceptual aliasing, that causes transitions to wrong actions or entry points. Furthermore, the interpolation to the entry point does not guarantee a collision-free trajectory. In CIP, a transition between actions only happens in *decision states*. This limits the number of possible successor actions to the intended ones for a situation and thereby avoids perceptual aliasing and wrong transitions. This guarantees a successful transition between actions when reproducing known situations.

6.6.2 Subjective Data

From analyzing the questionnaires and the responses in the interviews can be concluded, that SBP is an easily usable and intuitive framework for programming a-priori known tasks and conditions. However, compared to SBP, CIP has the advantage that overlapping parts of actions can be reused between different scenarios and complex tasks can be incrementally generated. For tasks with several different decisions and actions, it is difficult to predict all scenarios and to demonstrate the corresponding behavior prior to the execution. We argue, that for more complex tasks, the advantages of CIP can be fully exploited, since the user does not have to anticipate or detect new situations, but can demonstrate new actions when anomalies are detected during the execution. Furthermore, the combined teaching and execution of CIP gives the users the opportunity to instantly verify the result of their demonstrations.

From analysis of the NASA-TLX sub-categories can be seen, that CIP especially reduces the user's mental workload when programming a task, compared to UIP. CIP reached in this sub-category a score of 6.2, compared to 8.5 for UIP. This is in accordance with the results from the guided interviews, where 19 of 21 participants mentioned as advantages of CIP, that the robot autonomously detects new situations and that the user does not have to pay constant attention. Whereas the negative aspects for UIP, related to an increased need for attention and mental demand, were mentioned 17 times. The increased intuitiveness of CIP over UIP was confirmed by the overall evaluation of the methods. As mentioned 11 times in the interviews, deciding for the right moment to stop the task execution of UIP in order to add a new action is not intuitive for the user. This decision requires a deeper understanding of the principle behind the method. As seen in Fig. 10, by eliminating the user's influence on that decision, the robot automatically stops the task execution when an anomaly can be sensed by the robot, which significantly improves the task performance.

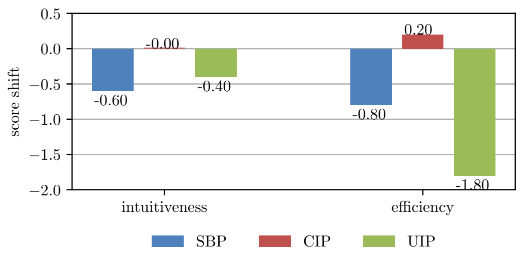


Fig. 19 Shift of *intuitiveness* and *efficiency* scores before and after users have seen the robot's execution, i.e. a negative sign means that users have downgraded their ratings on average compared to their first ratings.

6.7 Post-Experiment User Ratings

In a final evaluation, we consulted five subjects[3] again to obtain their ratings for the intuitiveness and efficiency of each of the programming approaches. Since we have their ratings from before seeing the execution, we are able to compare their scores from before and after they have seen the execution of their own programmed tasks. Fig. 19 shows the results of the comparison. We can see that for both alternative approaches, the intuitiveness and efficiency dropped noticeable, while for our approach the intuitiveness remained the same (no change) and the efficiency increased by 0.2 points on the Likert-type scale. That supports the assumption, that due to the additional feedback loop in CIP, the participants have a better understanding of the robot's changing task knowledge when teaching a task compared to SBP.

This concludes that our framework is more transparent to the user in terms of what the system has learned and what the robot is expected to do in the task execution. In relation to that, [23] proposed an objective metric to evaluate the teacher's efficiency in robot learning, given a specific feedback channel, e.g. by observing the robot's execution performance. We concluded from the success rate that our method performed best but the efficiency was still rated by the users, which is a subjective measure. Hence, analyzing the effect of different task representations used as feedback channel in terms of the teaching efficiency could aid developers to create better user interfaces.

7 Conclusion and Future Work

We presented a framework that allows non-experts to intuitively program conditional tasks that enables the robot to make decisions ad-hoc during task execution. Hereby, the complete task structure is transferred by demonstration involving sensor readings of motion, force and grasp status and no predefined symbols or objects are required. We have demonstrated that task decisions can be effectively transferred by our interactive programming scheme, where the robot asks for user input in unknown environmental situations. An on-line anomaly detection reduces the user's workload by just querying necessary information and guarantees a functioning task model, since transitions to specific actions are only allowed within a decision state. This enables the user to scale the complexity of a task over time without cumbersome reprogramming of the whole task.

We compared our framework experimentally with two alternative approaches in a user study, which lets us draw the conclusion that *Collaborative Incremental Programming* is the approach which users rated as most intuitive to use to transfer knowledge to the system and the one that is reliable in handling decisions during execution according to its success rate.

As a limitation, we state that the anomaly detection halts the robot motion in order to query the user, which could be infeasible for highly dynamic tasks. Further, the reusability of the knowledge represented as task-graph could be improved in the future as only the branching at decision states is considered, but not merging states or the recurrence of actions.

In this work, we stressed on the discrepancies in human and robot perception, since humans use vision but not every robotic system does so. In the future, we would like to consider a vision system as an additional sensor source given the fact that some anomalies can be observed visually before object interaction and some cannot, such as interaction forces.

[3] The robot's execution success was evaluated in absence of the 21 users. After that, we were able to contact again five subjects from the original group for this analysis.

Acknowledgements This work has been partially funded by the Helmholtz Association.

Conflict of interest

The authors declare that they have no conflict of interest.

References

1. Caccavale, R., Saveriano, M., Finzi, A., Lee, D.: Kinesthetic teaching and attentional supervision of structured tasks in human-robot interaction. Autonomous Robots (AURO) (2017)
2. Calinon, S., Lee, D.: Learning control. In: P. Vadakkepat, A. Goswami (eds.) Humanoid Robotics: a Reference. Springer (2018)

3. Di Lello, E., Klotzbucher, M., De Laet, T., Bruyninckx, H.: Bayesian time-series models for continuous fault detection and recognition in industrial robotic tasks. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 5827–5833. IEEE (2013)
4. Donald, B.R.: A geometric approach to error detection and recovery for robot motion planning with uncertainty. Artificial Intelligence **37**(1-3), 223–271 (1988)
5. Eiband, T., Saveriano, M., Lee, D.: Intuitive programming of conditional tasks by demonstration of multiple solutions. IEEE Robotics and Automation Letters **4**(4), 4483–4490 (2019). DOI 10.1109/LRA.2019.2935381
6. Eiband, T., Saveriano, M., Lee, D.: Learning haptic exploration schemes for adaptive task execution. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 7048–7054. IEEE (2019)
7. Grant, D.A.: The latin square principle in the design and analysis of psychological experiments. Psychological bulletin **45**(5), 427 (1948)
8. Hart, S.G., Staveland, L.E.: Development of nasa-tlx (task load index): Results of empirical and theoretical research. In: Advances in psychology, vol. 52, pp. 139–183. Elsevier (1988)
9. Khalastchi, E., Kalech, M.: A sensor-based approach for fault detection and diagnosis for robotic systems. Autonomous Robots **42**(6), 1231–1248 (2018)
10. Khalastchi, E., Kalech, M., Kaminka, G.A., Lin, R.: Online data-driven anomaly detection in autonomous robots. Knowledge and Information Systems **43**(3), 657–688 (2015)
11. Maeda, G., Ewerton, M., Osa, T., Busch, B., Peters, J.: Active incremental learning of robot movement primitives. In: Conference on Robot Learning, pp. 37–46 (2017)
12. Materna, Z., Kapinus, M., Beran, V., Smrž, P., Zemčík, P.: Interactive spatial augmented reality in collaborative robot programming: User experience evaluation. In: 2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), pp. 80–87. IEEE (2018)
13. Mollard, Y., Munzer, T., Baisero, A., Toussaint, M., Lopes, M.: Robot programming from demonstration, feedback and transfer. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1825–1831. IEEE (2015)
14. Muench, S., Kreuziger, J., Kaiser, M., Dillman, R.: Robot programming by demonstration (rpd)-using machine learning and user interaction methods for the development of easy and comfortable robot programming systems. In: Proceedings of the International Symposium on Industrial Robots, vol. 25, pp. 685–685. International Federation of Robotics & Robotic Industries (1994)
15. Naumann, A., Hurtienne, J.: Benchmarks for intuitive interaction with mobile devices. In: Proceedings of the 12th International Conference on Human Computer Interaction with Mobile Devices and Services, MobileHCI '10, p. 401–402. Association for Computing Machinery, New York, NY, USA (2010). DOI 10.1145/1851600.1851685. URL https://doi.org/10.1145/1851600.1851685
16. Niekum, S., Chitta, S., Barto, A.G., Marthi, B., Osentoski, S.: Incremental semantically grounded learning from demonstration. In: Robotics: Science and Systems, vol. 9. Berlin, Germany (2013)
17. Niekum, S., Osentoski, S., Konidaris, G., Chitta, S., Marthi, B., Barto, A.G.: Learning grounded finite-state representations from unstructured demonstrations. The International Journal of Robotics Research **34**(2), 131–157 (2015)
18. Pais, A.L., Umezawa, K., Nakamura, Y., Billard, A.: Task parameterization using continuous constraints extracted from human demonstrations. IEEE Transactions on Robotics **31**(6), 1458–1471 (2015)
19. Park, D., Erickson, Z., Bhattacharjee, T., Kemp, C.C.: Multimodal execution monitoring for anomaly detection during robot manipulation. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 407–414. IEEE (2016)
20. Pastor, P., Kalakrishnan, M., Chitta, S., Theodorou, E., Schaal, S.: Skill learning and task outcome prediction for manipulation. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 3828–3834. IEEE (2011)
21. Rodriguez, A., Bourne, D., Mason, M., Rossano, G.F., Wang, J.: Failure detection in assembly: Force signature analysis. In: Automation Science and Engineering (CASE), IEEE Conference on, pp. 210–215. IEEE (2010)
22. Sauer, L., Henrich, D., Martens, W.: Towards intuitive robot programming using finite state automata. In: Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz), pp. 290–298. Springer (2019)
23. Sena, A., Howard, M.: Quantifying teaching behavior in robot learning from demonstration. The International Journal of Robotics Research **39**(1), 54–72 (2020)
24. Steinmetz, F., Nitsch, V., Stulp, F.: Intuitive task-level programming by demonstration through semantic skill recognition. IEEE Robotics and Automation Letters **4**(4), 3742–3749 (2019). DOI 10.1109/LRA.2019.2928782
25. Su, Z., Kroemer, O., Loeb, G.E., Sukhatme, G.S., Schaal, S.: Learning manipulation graphs from demonstrations using multimodal sensory signals. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 2758–2765. IEEE (2018)
26. Willibald, C., Eiband, T., Lee, D.: Collaborative programming of conditional robot tasks. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), p. accepted (2020)