End-User Programming of Low- and High-Level Actions for Robotic Task Planning

Ying Siu Liang, Damien Pellier, Humbert Fiorino, and Sylvie Pesty

***Abstract*— Programming robots for general purpose applications is extremely challenging due to the great diversity of end-user tasks ranging from manufacturing environments to personal homes. Recent work has focused on enabling end-users to program robots using Programming by Demonstration. However, teaching robots new actions from scratch that can be reused for unseen tasks remains a difficult challenge and is generally left up to robotic experts. We propose iRoPro, an interactive Robot Programming framework that allows end-users to teach robots new actions from scratch and reuse them with a task planner. In this work we provide a system implementation on a two-armed Baxter robot that (i) allows simultaneous teaching of low- and high-level actions by demonstration, (ii) includes a user interface for action creation with condition inference and modification, and (iii) allows creating and solving previously unseen problems using a task planner for the robot to execute in real-time. We evaluate the generalisation power of the system on six benchmark tasks and show how taught actions can be easily reused for complex tasks. We further demonstrate its usability with a user study (N=21), where users completed eight tasks to teach the robot new actions that are reused with a task planner. The study demonstrates that users with any programming level and educational background can easily learn and use the system.**

I. INTRODUCTION

Despite the ongoing advances in Robotics and A.I., it is extremely challenging to pre-program robots for specific end-user applications. Instead of developing robots for domain-specific tasks, a more flexible solution is to have robots learn new actions directly from end-users and let them customise the robot for their specific application. Programming by Demonstration (PbD) [1] has been used to allow end-users to teach robots actions in an intuitive way by taking demonstrations as input and inferring a policy for the task. However, PbD solutions usually require users to teach robots an action sequence to achieve a certain goal. If the goal changes, the user has to teach the robot a new sequence.

Consider the Tower of Hanoi, a puzzle consisting of three pegs and a number of differently-sized disks, stacked on one peg in descending order, with the largest peg at the bottom. The goal is to move the entire stack from one peg to another, by moving one disk at a time, and only to a larger disk or an empty peg. The solution is different depending on the given number of disks. If we want to teach a robot to solve this problem, it would be infeasible to demonstrate the solution each time. A more efficient approach would be to teach the robot the primitive action of moving a disk, associate rules or *conditions* to this action (*e.g.,* smaller disks can only be placed on top of larger ones), and have the robot generate an optimal solution using a task planner [2].

Y.S. Liang, D. Pellier, H. Fiorino, and S. Pesty are with the Univ. Grenoble Alpes, LIG, F-38000 Grenoble, France {liangyi, pellierd, fiorinoh, pestys}@univ-grenoble-alpes.fr

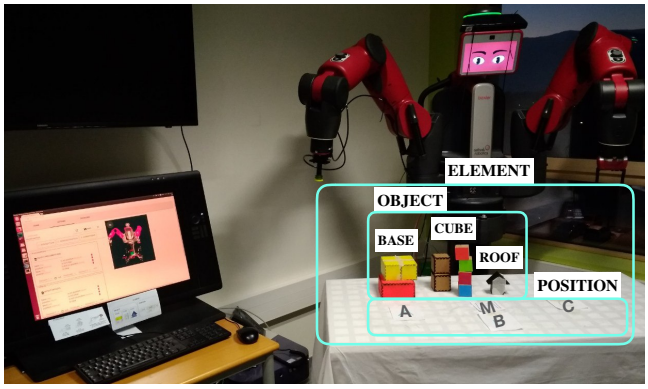

Fig. 1: Users programmed the Baxter robot via a graphical interface to manipulate objects (shown with their type hierarchies) in the task domain.

Our research argues for teaching robots primitive actions, instead of entire action sequences, and delegating the logical reasoning process of finding a solution to task planners. Even though task planners are generally used by domain-experts, we have previously shown that users with little to no programming experience can easily learn and use symbolic planning languages [3]. Based on the obtained results, this work presents iRoPro, an interactive Robot Programming system (Sec. III). It is a working end-to-end system that allows efficient programming of both *how* an action is performed (low-level) and *when* it can be applied (high-level), while generalising both aspects to previously unseen scenarios. We implement the system on a Baxter robot and developed a graphical interface that allows users to teach new actions by kinesthetic demonstration, modify their conditions and define previously unseen problems to solve with a task planner (Sec. IV). We demonstrate our system's capability to generalise primitive actions on six benchmark tasks that are programmed and executed on the robot (Sec. V). We empirically investigate the system's usability and validate its intuitiveness through a study with users of different educational backgrounds and programming levels (Sec. VI). To better understand user teaching strategies, we split participants into two control groups, with and without automatic condition inference, and showed that users in both groups can easily learn and use the system. Finally, we discuss limitations and possible extensions to further increase the system's generalisability (Sec. VII).

II. RELATED WORK

Our work relates to several topics explored in previous research. End-user robot programming has been addressed previously for industrial robots to be programmed by non-robotics domain experts, where users specify and modify existing plans for robots to adapt to new scenarios [4]. In our work we argue for the use of task planners to automatically generate plans for new scenarios.

Previous work has addressed knowledge engineering tools for constructing planning domains but usually require PDDL experts or common knowledge in software engineering (*e.g.,* itSIMPLE [5]). There has been previous work on integrating task planning with robotic systems [6] and learning preconditions and effects of actions to be used in planning [7]. However, the robot is usually provided with a fixed set of low-level motor skills. We do not provide the robot with any predefined actions but allow users to teach both low- and high-level actions from scratch.

Programming by demonstration [1] has been commonly applied to allow end-users to teach robots new actions by demonstration. Recent work has focused on mobile and industrial manipulators [4] and learning from single demonstrations [8]. Alexandrova et al. created an end-user programming framework with an interactive action visualisation allowing the user to teach new actions from single demonstrations but do not reuse them with a task planner.

Most closely related to our approach is the work by Abdo et al. [9] where manipulation actions are learned from kinesthetic demonstrations and reused with task planners. However, the approach requires 5-10 demonstrations to learn action conditions which becomes tedious and impractical if several actions need to be taught. Our work argues for having the user act as the expert by letting them correct inferred action conditions, thus allowing a new action to be learned from a single demonstration. We further provide a graphical interface that allows users to create new actions and previously unseen problems that can be solved with task planners.

III. APPROACH

Our approach aims at providing end-users with an intuitive way of teaching robots new actions that can be reused with a task planner to solve more complex tasks. Given a single demonstration, the robot should learn both *how* (Sec. III-A) and *when* (Sec. III-B) an action should be applied. To accelerate the programming process, action conditions are directly inferred from a single demonstration (Sec. III-C). The action generalisation is performed on both low- and high-level representations (Sec. III-D), allowing it to be reused with a task planner (Sec. III-E). We will describe our approach in the following sections.

A. Low-level Action Representation

We represent low-level actions as proposed in previous work using keyframe-based PbD [8], where the action is represented as a sparse sequence of gripper states (open/close) and end-effector poses relative to perceived objects or to the robot's coordinate frame. During the demonstration phase the user guides the robot arm using kinesthetic manipulation and saves poses that they find relevant for the action. For example, the pick-and-place action of an object to a marked position could be represented as poses relative to the object (for the pick action), poses relative to the target position (for the place action), and corresponding open/close gripper states. Action executions are performed by first detecting the landmarks in the environment, calculating the end-effector poses relative to the observed landmarks, and interpolating between the poses.

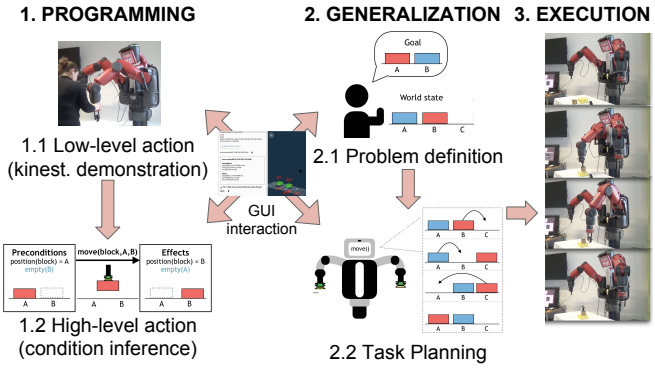

Fig. 2: Overview of iRoPro to teach low- and high-level actions: the user interacts with the GUI to run the demonstration, modify inferred action conditions and to create new planning problems for the robot to solve and execute.

While these actions can be learned from multiple demonstrations [10], we take the approach that only requires a single demonstration by heuristically assigning poses and letting the user correct them if needed [8]. Thus, the first demonstrated action is already an executable action. The user can teach multiple manipulation actions and discriminate between them by associating different conditions that specify *when* the robot should use them (*e.g.,* actions using claw or suction grippers). These conditions are discussed next in Sec. III-B.

B. High-level Action Representation

We represent high-level actions similar to previous work on task planning [2], where an action is represented as a tuple $a = (\text{param}(a), \text{pre}(a), \text{eff}(a))$, whose elements are:

- $\text{param}(a)$: set of parameters that a applies to
- $\text{pre}(a)$: set of predicates that must be true to apply a
- $\text{eff}(a)^-$: set of predicates that are false after applying a
- $\text{eff}(a)^+$: set of predicates that are true after applying a

where $\text{eff}(a) = \text{eff}(a)^- \cup \text{eff}(a)^+$. Action parameters are world instances that the robot interacts with and are associated with a *type*. For example, in iRoPro, we implemented a type hierarchy, consisting of a general type ELEMENT, divided into POSITION and OBJECT, which further divides into BASE, CUBE, and ROOF (Fig. 1).

Predicates are used to describe object states and relations between them and are defined in first-order logic. In our

graphical interface, predicates are translated from first-order logic ('on(obj, A)') to English statements ('obj is on A').

In iRoPro, we implemented predicates that are commonly used in task planning domains as well as two additional ones to describe object properties:

- *ELEMENT is clear*: an element has nothing on top of it
- *OBJECT is on ELEMENT*: an object is on an element
- *OBJECT is stackable on ELEMENT*: an object can be placed on an element
- *OBJECT is flat*: an object has a flat top
- *OBJECT is thin*: an object is thin

We assume that CUBE and BASE objects are flat, while CUBE and ROOF objects are thin enough for the robot to grasp. The set of inferred types and predicates could be extended for more complex tasks (*e.g.,* object colour or orientation [11]).

C. Action Inference from Demonstration

Instead of manually defining action parameters, preconditions, and effects, we accelerate the programming process by inferring them from the observed sensor data during the teaching phase. Object types are inferred based on their detected bounding boxes (see Sec. IV-A). Object positions are determined by the proximity of the object to given positions. If the nearest position p to the object o is within a certain threshold d, then the predicates 'o is on p' and 'p is not clear' are added to the detected world state.

To infer action conditions, the robot perceives the initial world state before and after the kinesthetic action demonstration as seen in similar work for learning object manipulation tasks [12]. Let $O_1 = \{\phi_1, \phi_2, ...\}$ be the set of predicates observed before the action demonstration and $O_2 = \{\psi_1, \psi_2, ...\}$ after. The action inference is the heuristic deduction of predicates that have changed between O_1 and O_2, *i.e.,*

$$\text{pre}(a) = (O_1 - O_1 \cap O_2) = \{\phi_i | \phi_i \in O_1 \wedge \phi_i \notin O_2\},$$
$$\text{eff}(a) = (O_2 - O_1 \cap O_2) = \{\psi_i | \psi_i \notin O_1 \wedge \psi_i \in O_2\},$$

where $\text{eff}(a)$ includes positive and negative effects (Fig. 3). A predicate ϕ has variables $\text{var}(\phi) = \{v_1, v_2, \dots\}$, where each v_i has a type. Therefore, action parameters are the set of variables that appear in either preconditions or effects, *i.e.,*

$$\text{param}(a) = \{v_i | \quad \exists \phi \in \text{pre}(a) \text{ s.t. } v_i \in \text{var}(\phi) \\ \vee \exists \psi \in \text{eff}(a) \text{ s.t. } v_i \in \text{var}(\psi)\}.$$

Note that conditions could be learned from multiple demonstrations [9], [7]. Our work argues for accelerating the teaching phase by learning from a single demonstration and letting the user act as the expert to correct wrongly inferred conditions.

D. Action Generalisation

The low-level representation (Sec. III-A) generalises motion trajectories by re-calculating poses based on detected landmarks from the demonstrated to the new environment.

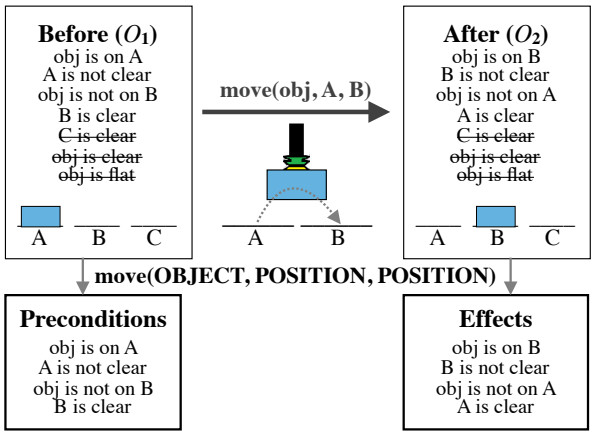

Fig. 3: Example of a high-level action for moving an object from A to B. Conditions are inferred from the observed predicates before (O_1) and after (O_2) the demonstration.

The high-level representation (Sec. III-B) specifies when an action can be applied, therefore allows taught low-level motion trajectories to be reused for other objects (*e.g.,* use suction grip for all objects, regardless of their dimension) or to be restricted for certain types (*e.g.,* only BASE objects). By combining these two representation levels, taught actions can be generalised for more complex environments and the user can customise them for their specific use case.

E. Task Planning

Task planners are used to generate solutions, or action sequences, to solve complex problems. Given a description of a planning *domain*, *i.e.,* object types, actions with preconditions and effects, we can define a planning *problem* with an initial state and a desired goal state. The planner generates an optimal action sequence, or *plan*, which guarantees the transition from initial state to the goal state. PDDL [13] is often used as a standard encoding language for planning problems. A move action as shown in Fig. 3 is defined as follows:

```
(:action move
 :parameters (?obj - object
              ?A - position ?B - position)
 :precondition (and (on ?obj ?A)(clear ?B)
            not(on ?obj ?B) not(clear ?A))
 :effect (and (on ?obj ?B) (clear ?A)
           not(on ?obj ?A) not(clear ?B))
```

A planning problem consists of an initial state and a goal state and can be solved using existing actions in the domain. For example, the problem of swapping two objects obj1, obj2 on A and B respectively with C unoccupied, can be defined as:

```
(:objects obj1 obj2 - object
          A B C - position)
(:init (and (on obj1 A) (on obj2 B)
            (clear C))
(:goal (and (on obj1 B) (on obj2 A)))
```

The planner would generate the following action sequence:

```
1. move(obj1, A, C)
2. move(obj2, B, A)
3. move(obj1, C, B)
```

IV. System

A. Platform & Implementation Details

We implemented our system on a Baxter robot with two arms (one claw and one suction gripper), both with 7-DoF and a load capacity of 2.2kg. For the object perception we mounted a Kinect Xbox 360 depth camera on the robot. We developed a user interface as a web application that can be accessed via a browser on a PC, tablet or smartphone. The source code for iRoPro is developed in ROS [14] and available online[1]. The low-level action is learned using the open-source system Rapid PbD[2]. The integration of the task planner is implemented using the ROS package PDDL planner[3].

In our implementation, landmarks are either predefined table positions or objects that are detected from Kinect point cloud clusters using an open-source tabletop segmentation library[4]. An object $obj = (x, y, z, width, length, height)$ is represented by its detected location and bounding box, which are used to infer its type and related predicates (Sec. III-C) The user creates a complete PDDL domain via the GUI by teaching new actions and problems that can be solved with the integrated task planner.[5]

B. Interactive Robot Programming

The user interacts with the GUI (Fig. 4) to visualise the robot and the detected objects, create new actions, run the kinesthetic teaching by demonstration, modify inferred types or predicates, create and solve new problems with the task planner. The interactive robot programming cycle consists of creating and modifying actions and problems.

a) Actions: New actions are taught by kinesthetically moving the robot's arms, where both low-level and high-level actions are learned and generalised. The low-level action is learned by keyframe-based demonstration (Sec. III-A). To verify the taught action, the user can have the robot re-execute it immediately. The high-level action is inferred automatically by capturing the world state before and after the action demonstration (as described in Sec. III-C). The user can modify the action properties if the inference was not correct. To teach more actions, the user can either create a new one or copy a previously taught action and modify it.

b) Problems: New planning problems can be created if at least one action exists. To create a problem, the robot first detects the existing landmarks and infers their types and initial states. The user can modify them if the inference was not correct. Then, the user enters predicates that describe the goal states to achieve. The complete planning domain and problem are translated into PDDL and sent to the Fast-Forward planner [15]. If a solution is found that reaches the goal, it is displayed on the GUI for the user to verify and execute on the robot. If no solution is found or if the generated plan is wrong, the user can open a debug menu which summarises the entire planning domain with hints described in natural language to investigate the problem (*e.g.,* 'make sure the action effects can achieve the goal states'). In our user study (Sec. VI) we found that this helped users understand how the system worked and why the generated plan was wrong. Once the user modified actions, initial or goal states, they can relaunch the planner to see if a correct plan is generated. To solve new tasks, the user can create a new problem or modify existing ones by redetecting the objects.

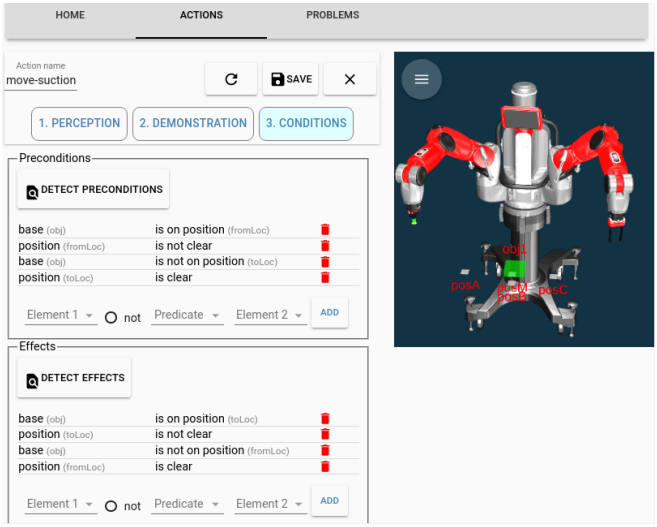

Fig. 4: The iRoPro interface showing the action condition menu and an interactive visualisation of the Baxter robot and detected objects.

C. Plan Execution

The generated plan is a sequence of actions with parameters that correspond to detected objects. For each action, the sequence of end-effector poses are calculated relative to the landmarks that the action is being applied to (Sec. III-A). To accelerate the execution, we only detect the landmarks once at the start and save their new positions in a mental model. After each action execution, the user can confirm that it executed correctly and the mental model is updated with the latest positions of the changed landmarks. The mental model is also used as a workaround for our limited perception system for problems with stacked objects in their initial states (Sec. VII).

V. System Evaluation

We evaluate our system's generalisability on six benchmark tasks (Table I) and show how taught primitive actions can be reused for complex tasks. The tasks involved manipulating different object types on four marked positions with

[1] https://github.com/ysl208/iRoPro/tree/cond
[2] https://github.com/jstnhuang/rapid_pbd
[3] http://docs.ros.org/indigo/api/pddl_planner
[4] https://github.com/jstnhuang/surface_perception
[5] Video can be viewed at https://youtu.be/YCDrC0UFX38

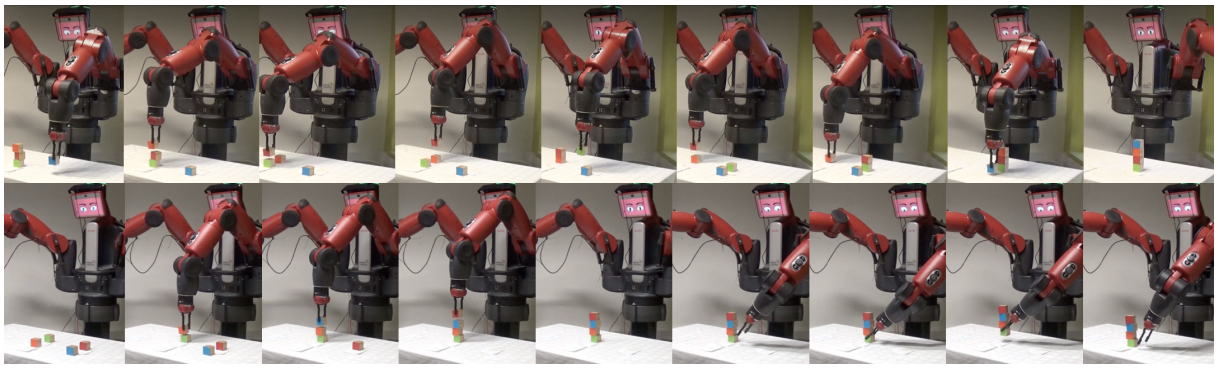

Fig. 5: Snapshots from the executions of the system evaluation showing Task 3 (top) and 4 (bottom).

both claw and suction grippers. We take the Blocksworld domain [16] for building and rebuilding stacked objects (Tasks 1-4) and an elaborate version of the Tower of Hanoi problem with different object types to build a 'house' (Tasks 5&6). Instead of disks, we decided to use different object types (ROOF, CUBE, BASE), where BASE corresponds to the largest disk, followed by CUBE and ROOF. The rules for stacking different objects still apply (*e.g.,* BASE cannot be stacked on top of ROOF or CUBE). However, to demonstrate the generalisation of actions to diverse tasks, additional constraints are added as objects cannot be all manipulated in the same way. The order of the tasks was given with increasing complexity, requiring the user to modify existing actions or teach new actions from scratch.

A. Protocol

The tasks were programmed by one of the authors, with the most efficient teaching strategy of minimising the number of actions created and generalising them by changing the action properties (as described in Sec. III-D). Depending on the given task and involved objects, the experimenter decided what manipulation action needed to be taught. Only one planning problem was created and reused for all tasks by redetecting the objects in the initial state and changing the goal state. When the generated plan was incorrect, the debug menu on the GUI was used to determine the changes to be made to generalise the actions. A task was considered completed when the generated plan was correct and the robot successfully executed it. As the mental model saved the latest object positions after an action execution, Tasks 3 and 6 were continued from the preceding tasks and did not require redetecting the initial states.

B. Results

We programmed three manipulation actions for the six benchmark tasks, which involved demonstrating pick-and-place actions with claw and suction grippers from the *top* and from the *side*. Actions were generalised by changing parameter types (*e.g.,* from CUBE or POSITION to ELEMENT) or adding preconditions or effects which were not inferred automatically. For pick-and-place actions from the top, 'obj is clear' was added as a precondition (Task 1-3), while it was not included when picking an object from the side to allow moving a pile of objects (Task 4). For actions involving the claw gripper, the precondition 'obj is thin' was added so that the robot would only use it on ROOF and CUBE objects, similarly 'is flat' for the suction gripper (Task 5&6). The 'is stackable' condition was used for the Tower of Hanoi as an equivalent to the rule 'larger objects cannot be placed on top of smaller ones'. Due to the noisy sensing and control of the Baxter robot, action executions failed occasionally, even though the generated plan was correct. Overall, the robot was able to generate plans for all tasks and executed them at least twice (Fig. 5). While the experimental scope was limited and set in a controlled environment, it still demonstrates iRoPro's expressivity. With minimal end-user programming effort, manipulation actions can be taught from scratch and reused for a diverse range of tasks, even beyond the six benchmark tasks.

TABLE I: Benchmark tasks for the system evaluation. Three different pick-and-place actions were programmed.

#	Task goal	Pick-and-place action
1	Build tower with 3 CUBES	claw from top
2	Build tower with 4 CUBES	claw from top
3	Rebuild Task 2 on a different position	claw from top
4	Build tower and move (w/o disassembly)	claw from top & side
5	Build house with BASE, CUBE, ROOF	claw & suction from top
6	Rebuild Task 5 on a different position	claw & suction from top

VI. USER EVALUATION

The second part of the evaluation was conducted using THEDRE [17], a human experiment design method that combines qualitative and quantitative approaches to continuously improve and evaluate the developed system from the experimental ground. The aim was to evaluate our approach with real end-users and we were also interested in the user's programming strategy for using the system. We split participants into two control groups, with and without condition inference (Sec. III-C) and evaluated user performance in terms of programming times for completing a set of benchmark tasks. We set the following hypotheses for our experiments:

H1 Action creation: users can teach new low- and high-level actions by demonstration

H2 Problem solving: users can solve new problems by defining the goal states and executing the plan on Baxter
H3 Autonomous system navigation: users understand the system and can navigate and troubleshoot on their own
H4 Condition inference (CI) - Group 1 vs 2: users without CI will understand the system better
H5 Pre-study test (PT): users that score higher in the PT have shorter programming times

A. Participants

The study was conducted with 21 participants (10M, 11F) in the range of 18-39 years (M=24.67, SD=6.1). We recruited participants with different educational background and programming levels: 6 'CS' (either completed a degree in computer science or were currently pursuing one), 7 'non-CS' (have previously taken a programming course before), and 8 'no experience' (only had experience with office productivity software). Furthermore, 3 participants (in 'CS') have programmed a robot before, out of which 1 had intermediate experience with symbolic planning languages while the remaining participants had no experience in either. One participant in the category 'non-CS' failed to complete the majority of tasks and was excluded from the result evaluation. The two control groups included equal number of participants in all three categories.

B. Protocol

Users were first given a brief introduction to task planning concepts, the Baxter robot and the experimental set up (Fig. 1). They were then asked to complete a pre-study test to capture the participant's understanding of the presented concepts. Users were given 8 tasks to complete, where the first two were practice tasks to introduce them to the system (Table II). The tasks were designed to address different aspects to familiarise them with the system: create new actions (Task 6), modify parameter types (Tasks 4&7), modify action conditions (Tasks 3,5,8). For each task they needed to create a new problem, define the goal states, and launch the planner to generate an action sequence. When the generated plan was correct, they were executed on the robot. Otherwise, the user had to modify the existing input until the plan was correctly generated. Tasks 6-8 were similar to the previous tasks (1-5) but use both robot grippers.

C. Metrics

We captured the following data during the experiments:

1) **Qualitative data:** video recording of the experiment, observations during the experimental protocol.
2) **Quantitative data:** task duration and UI activity log, pre-study test, post-study survey.

The pre-study test included 7 questions related to their understanding of the concepts presented at the start of the experiment, *e.g.,* syntax ('If move(CUBE) describes a move action, tick all statements that are true.'), logical reasoning ('Which two conditions can never be true at the same time?'), and other concepts ('Tick all predicates that are required as preconditions for the given action'). The questions were multiple choice and the highest achievable score was 7.

In the post-study survey we used the System Usability Scale (SUS) [18] where participants had to give a rating on a 5-Point Likert scale ranging from 'Strongly agree' to 'Strongly disagree'. It enabled us to measure the perceived usability of the system with a small sample of users. As a benchmark, we compare overall responses to our previous user study [3], where users were simulated a similar robot programming experience using the Wizard-of-Oz technique but had no direct interaction with a working system. Finally, participants were asked which aspects they found most useful, most difficult, and which they liked the best and the least.

TABLE II: Benchmark tasks for the user study where the first two tasks were used to introduce participants to the system.

# Task description	Main solution
(1) move BASE object (suction grip)	create new action (+demo)
(2) move BASE object to any position	create new problem
3 swap two BASE objects	add condition ('is clear')
4 stack CUBE on BASE	modify types ('OBJECT')
5 do not stack CUBE on ROOF	add condition ('is stackable')
6 move ROOF object (claw grip)	create new action (+demo)
7 stack ROOF on a CUBE	modify types ('ELEMENT')
8 build a house (BASE, CUBE, ROOF)	navigate autonomously

TABLE III: User performance comparing task completion times with pre-study test scores.

	Main tasks (in min)		**PT score (out of 7)**	
	AVG	STD	AVG	STD
no experience	43.6	5.37	5.8	0.95
non-CS	36.6	7.46	6.2	0.55
CS	43.8	14.13	5.3	1.11
Overall	**41.2**	**9.08**	**5.8**	**0.91**
Group 1	41.0	7.89	6.1	0.77
Group 2	41.4	10.56	5.5	0.98

D. Results

20 participants completed all tasks, while one 'non-CS' user failed to complete the majority of tasks and did not seem to understand the presented concepts. This participant was excluded in the results presented below (Table III):

H1)-H3) User performance: Users took between 22-60 minutes to complete the main tasks (3-8), with an average of 41.2 minutes. 'non-CS' users completed the tasks the fastest, followed by users with no programming experience. 'CS' users took on average longer as they were often interested in testing the system's functionalities that were beyond the given tasks.

Users initially had problems with different concepts that were presented at the start of the study, in particular they confused action parameters, preconditions and goal states. For example, in Task 3, 6 (or 30%) users tried to add intermediate action steps to achieve the goal state, instead of simply letting the planner generate the solution. In Task 4, 14 (or 70%) wanted to create a new action, even though they

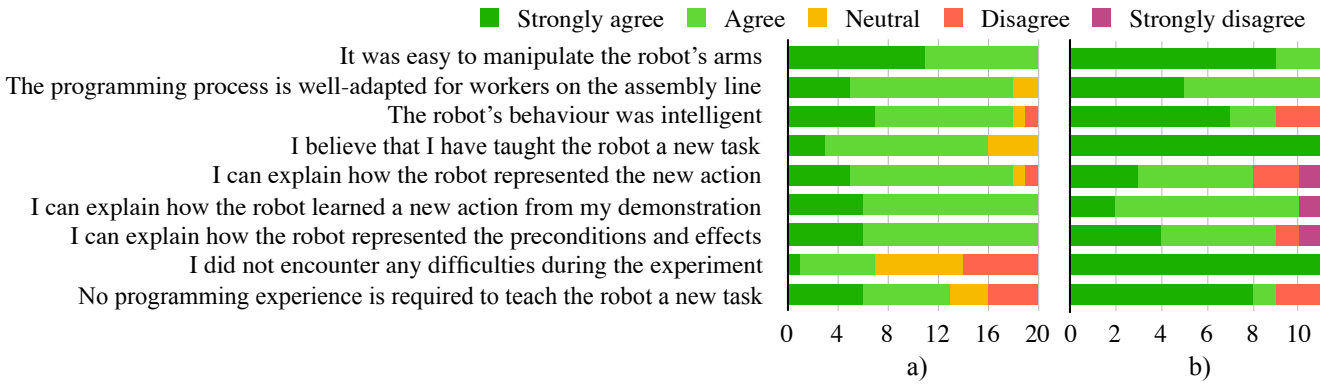

Fig. 6: User responses from the post-study survey comparing a) iRoPro (N=20) to b) our previous user study (N=11) [3]

could reuse the existing action by modifying the parameter types. However, by Task 6, all users were able to use the system autonomously to create new actions and problems and navigated the system with little to no guidance. By the end of the experiment, users programmed two manipulation actions (one for each gripper) that were reused to complete all 8 benchmark tasks.

H4) Condition inference (CI): We noticed a discrepancy in the programming strategies between the two control groups (Group 1 with CI vs. Group 2 without CI). Participants in Group 1 had the tendency to leave the inferred conditions unmodified without adding conditions that would improve the action's generalisability to different use cases. As participants in Group 2 had to add action conditions manually, they considered all predicates they deemed necessary for the action and added additional ones that were required for later tasks. Thus, Group 2 took on average longer to complete tasks where a new action had to be created (Tasks 1&6), but was faster than Group 1 for subsequent tasks, where conditions had to be modified (Tasks 3,5,7). Overall both groups had similar completion times for all tasks.

H5) Pre-study test: As expected, participants who demonstrated a better understanding of the introduced concepts in the pre-study test completed the main tasks (Tasks 3-8) faster on average (p-value< 0.05). Users scored between 4.3-6.93 out of 7 points. 'non-CS' users scored above average points and completed the fastest. As an outlier we observed that the fastest participant scored only 4.7, but easily learned how to use the system and completed the tasks in 22 minutes. Even though Group 1 performed slightly better in the pre-study test than Group 2, both completion times were on average similar.

System usability and learnability: There are several ways to interpret the System Usability Scale (SUS) scores [18] obtained from the post-study survey. Using Bangor et al.'s categories [19], 14 (70%) users ranked iRoPro as 'acceptable', 6 (30%) rated it 'marginally acceptable', and no one ranked it 'not acceptable'. Correlating this with the Net Promoter Score, this corresponds to 10 (50%) participants being 'promoters' (most likely to recommend the system), 5 (25%) 'passive', and 5 (25%) 'detractors' (likely to discourage). Overall, iRoPro was rated with a good system usability and learnability.

Overall user experience: We compare responses to our previous user study (N=11) [3], where users had no direct interaction with the robot programming system as it was simulated using the Wizard-of-Oz technique. The main differences were noted regarding difficulties encountered during the experiment (Fig. 6): In our previous study we had 11 (or 100%) agree that they encountered no difficulties, while this time only 7 (or 35%) of our users stated the same. However, all of our users claimed to have a good understanding of the action representation and how the robot learned new actions from their demonstrations, while an average of 2 (18%) disagreed in [3]. Both differences can be explained by the fact that in this study, users had to use an end-to-end system to program the robot, while in our previous work users had no direct interaction with a working system. Even though our users encountered more difficulties, they got a better understanding of the functionalities due to getting hands-on experience. This also correlates with negative responses in our survey to the question if 'no programming experience was required' where 13/20 (65%) agreed and 4 disagreed. Overall, our user study received positive responses similar to the previous study.

9 (45%) users stated 'generate solutions to defined goals automatically' as the most useful feature, followed by 'robot learns action from my demonstration' (4 or 20%) – two main aspects of our approach. 4 (20%) stated the most difficult part as 'finding out why Baxter didn't solve a problem correctly', similarly 8 (40%) stated difficulties related to 'understanding predicates and defining conditions'. 11 (55%) disliked 'assigning action conditions' the most, while the rest stated different aspects. A common feedback was 'it takes time to understand how the system works at the start'. The most liked parts were 'executing the generated plan' (8 or 40%) and 'demonstrating an action on Baxter' (7 or 35%).

VII. DISCUSSIONS

In our evaluation scenarios we could have programmed more complicated manipulation actions such as turning or pushing for packaging tasks (as done previously in [20]). We decided to stick to simple pick-and-place manipulation actions, as our main focus was to evaluate iRoPro's usability

with end-users. Both system and user evaluations demonstrated that the proposed robot programming process for manipulation tasks can be learned easily by users with or without programming experience.

The workaround based on mental models of the environment only works if the environment is static and no external entity interferes with the world. The next step would be to move from controlled environments to more dynamic ones, whereby all relevant aspects of the environment need to be fully perceivable by the system. An improved perception system would also allow tracking and verifying action executions in case of failures in more complex environments.

Due to the Baxter robot's different grippers, we did not program actions that use both arms simultaneously (*e.g.,* for carrying a tray). A possible extension would be to include a better motion and task planning system to allow this while also considering self-collision avoidance. Furthermore, we did not program human-robot collaborative tasks, such as human-robot hand-over tasks. To allow this, more complex planning domains and better multi-modal communication would need to be implemented.

VIII. CONCLUSION

In this work we presented iRoPro, an interactive Robot Programming system that allows simultaneous teaching of low- and high-level representations of actions by demonstration. The robot reuses the actions with a task planner to generate solutions to previously unseen tasks that are more complex than the demonstrated action. The approach was implemented on a Baxter robot and we showed its generalisability on six benchmark tasks by teaching a minimal set of primitive actions that were reused for all tasks. We further demonstrated its usability with a user study (20 successful results, 1 unsuccessful result) where participants with diverse educational backgrounds and programming levels learned how to use the system in less than an hour. Both user performance and feedback confirmed iRoPro's usability, with the majority ranking it as 'acceptable' and half being promoters. Overall, we demonstrated that our approach allows users with any programming level to efficiently teach robots new actions that can be reused for complex manipulation tasks.

Future work will focus on exploring more challenging domains to extend the work to other platforms by including a wider range of predicates and probabilistic techniques to improve the condition inference. As we focused on controlled environments, further studies will involve more complex environments with factory workers who may ultimately use this technology.

ACKNOWLEDGMENTS

We would like to thank Nadine Mandran for supporting the experiment design process.

REFERENCES

[1] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Robot programming by demonstration," in *Handbook of Robotics*. Springer, 2008, pp. 1371–1394.

[2] M. Ghallab, D. Nau, and P. Traverso, *Automated planning: Theory & Practice*. Elsevier, 2004.

[3] Y. S. Liang, D. Pellier, H. Fiorino, and S. Pesty, "Evaluation of a Robot Programming Framework for Non-Experts using Symbolic Planning Representations," in *Intl. Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2017.

[4] M. Stenmark, M. Haage, and E. A. Topp, "Simplified programming of re-usable skills on a safe industrial robot: Prototype and evaluation," in *Intl. Conf. on Human-Robot Interaction*. ACM, 2017, pp. 463–472.

[5] T. S. Vaquero, J. R. Silva, F. Tonidandel, and J. C. Beck, "itSIMPLE: towards an integrated design system for real planning applications," *The Knowledge Engineering Review*, vol. 28, no. 2, pp. 215–230, 2013.

[6] M. Cashmore and M. Fox, "ROSPlan: Planning in the Robot Operating System," in *Intl. Conf. on Automated Planning and Scheduling*, 2015.

[7] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, "From skills to symbols: Learning symbolic representations for abstract high-level planning," *Journal of Artificial Intelligence Research*, vol. 61, pp. 215–289, 2018.

[8] S. Alexandrova, M. Cakmak, K. Hsiao, and L. Takayama, "Robot programming by demonstration with interactive action visualizations," in *Robotics: Science and Systems (RSS)*, 2014.

[9] N. Abdo, H. Kretzschmar, L. Spinello, and C. Stachniss, "Learning manipulation actions from a few demonstrations," in *Intl. Conf. on Robotics and Automation (ICRA)*. IEEE, 2013, pp. 1268–1275.

[10] S. Niekum, S. Osentoski, G. Konidaris, and A. G. Barto, "Learning and generalization of complex tasks from unstructured demonstrations," in *Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2012, pp. 5239–5246.

[11] C. Li and D. Berenson, "Learning object orientation constraints and guiding constraints for narrow passages from one demonstration," in *Intl. symposium on experimental robotics*. Springer, 2016, pp. 197–210.

[12] S. R. Ahmadzadeh, A. Paikan, F. Mastrogiovanni, L. Natale, P. Kormushev, and D. G. Caldwell, "Learning symbolic representations of actions from human demonstrations," in *Intl. Conf. on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3801–3808.

[13] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL-the planning domain definition language," 1998.

[14] M. Quigley, J. Faust, T. Foote, and J. Leibs, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, 2009.

[15] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *Journal of Artificial Intelligence Research*, pp. 253–302, 2001.

[16] J. Slaney and S. Thiébaux, "Blocks world revisited," *Artificial Intelligence*, vol. 125, pp. 119–153, 2001.

[17] N. Mandran and S. Dupuy-Chessa, "THEDRE: a Traceable Process for High Quality in Human Centred Computer Science Research," in *Intl. Conf. of System Development*, 2017, p. vol. 9.

[18] J. Brooke, "SUS: a retrospective," *Journal of usability studies*, vol. 8, no. 2, pp. 29–40, 2013.

[19] A. Bangor, P. T. Kortum, and J. T. Miller, "An empirical evaluation of the system usability scale," *Intl. Journal of Human-Computer Interaction*, vol. 24, pp. 574–594, 2008.

[20] Y. S. Liang, D. Pellier, P. S. Fiorino, Humbert, and M. Cakmak, "Simultaneous End-User Programming of Goals and Actions for Robotic Shelf Organization," in *Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018.