International Journal of Social Robotics (2022) 14:177–191
https://doi.org/10.1007/s12369-021-00775-9

iRoPro: An interactive Robot Programming Framework

Ying Siu Liang[1] · Damien Pellier[1] · Humbert Fiorino[1] · Sylvie Pesty[1]

Accepted: 6 March 2021 / Published online: 5 April 2021
© The Author(s), under exclusive licence to Springer Nature B.V. 2021

Abstract
The great diversity of end-user tasks ranging from manufacturing environments to personal homes makes pre-programming robots for general purpose applications extremely challenging. In fact, teaching robots new actions from scratch that can be reused for previously unseen tasks remains a difficult challenge and is generally left up to robotics experts. In this work, we present iRoPro, an interactive Robot Programming framework that allows end-users with little to no technical background to teach a robot new reusable actions. We combine Programming by Demonstration and Automated Planning techniques to allow the user to construct the robot's knowledge base by teaching new actions by kinesthetic demonstration. The actions are generalised and reused with a task planner to solve previously unseen problems defined by the user. We implement iRoPro as an end-to-end system on a Baxter Research Robot to simultaneously teach low- and high-level actions by demonstration that the user can customise via a Graphical User Interface to adapt to their specific use case. To evaluate the feasibility of our approach, we first conducted pre-design experiments to better understand the user's adoption of involved concepts and the proposed robot programming process. We compare results with post-design experiments, where we conducted a user study to validate the usability of our approach with real end-users. Overall, we showed that users with different programming levels and educational backgrounds can easily learn and use iRoPro and its robot programming process.

Keywords End-User robot programming · Programming by demonstration · Automated planning · Human-Robot interaction

1 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 them learn new actions directly from end-users who can customise the robot for their specific application. Programming by Demonstration (PbD) [8] is a popular approach for 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.

Different approaches to generate robot actions have been proposed [11]. The *reactive* approach is characterised by independent and concurrent basic behaviours that generate the robot global behaviour. The robot sensors are the input of the basic behaviours that, in turn, activate/inhibit the robot end effectors. This approach is effective but predicting the robot global behaviour is difficult since no explicit action

✉ Ying Siu Liang
yingsiu@liang.at

Damien Pellier
pellierd@univ-grenoble-alpes.fr

Humbert Fiorino
fiorinhu@univ-grenoble-alpes.fr

Sylvie Pesty
pestys@univ-grenoble-alpes.fr

[1] Université Grenoble Alpes, LIG, 38000 Grenoble, France

model is used. The *deliberative* approach consists of three steps: perception of the world state, plan making and action execution in order to change the world state. Plan making also known as Automated Planning [18], is based on an explicit action model encoded in a symbolic planning language. This action model is used by a task planner to compute action sequences achieving the expected robot behaviours.

Our research argues for teaching robots primitive actions, instead of entire action sequences, and delegating the logical reasoning process of finding a solution to a task planner. To this end, we present iRoPro, an interactive Robot Programming framework that allows efficient programming of both *how* an action is performed (low-level action representation) and *when* it can be applied (high-level), while generalising both aspects to new scenarios. The low-level representation allows the robot to execute the motion trajectory, while the high-level representation allows it to be used with a task planner.

We implement the framework on a Baxter robot and develop an intuitive graphical interface that allows users to teach new actions by demonstration and customise them to be reused for new problems that can be solved with a task planner. The developed end-to-end system involves solutions in perception (*e.g.,* object identification), motion planning (*e.g.,* manipulation, navigation, safety), cognitive robotics (*e.g.,* action learning, task planning) and human–robot interaction (*e.g.,* multi-modal interaction and teacher feedback). Even though task planners are generally used by domain-experts, we show that users with little to no programming experience can easily learn and use their main concepts. We conduct pre-design experiments, where we simulate the framework with the Wizard-of-Oz technique to evaluate the usability of the programming process. We compare these results with post-design experiments and empirically investigate our system's usability with a user study (N=21) where real end-users programmed a Baxter robot directly with our end-to-end system.

In Sect. 2 we give a brief overview of PbD and Automated Planning, the two underlying techniques of the framework, and discuss related work in Sect. 3. In Sect. 4 we present iRoPro, the interactive Robot Programming framework and its main components. Then in Sect. 5 we provide details of the system implemented on a Baxter robot and the user programming process. Section 6 presents the experimental evaluation of our approach, where we compare pre-design experiments with post-design experiments. Finally, in Sect. 7, we conclude by discussing limitations and possible extensions to further increase the system's generalisability Sect. 9.

2 Background

Traditional robot programming processes have task-specific definitions, are generally robot-dependent and require programming expertise. In the past few decades, different techniques have been developed to facilitate the robot programming process. Biggs et al. [7] defined two main categories, manual and automatic programming, distinguishing between systems where users can or cannot directly control the robot's executed behaviour.

In manual programming, the user encodes the robot's behaviour via text-based systems using procedural languages such as python, C++, java or graphical systems that use a graph, flow-chart or diagram [14,25,31]. For automatic programming techniques, robots generate their behaviour from data provided as input to the system. We differentiate between Deep Learning (DL) [42], where the robot is provided a large amount of labelled or unlabelled data, Reinforcement Learning [19,22], where the robot gathers the data by exploring the environment, and Programming by Demonstration (PbD) [5,8], where the robot learns from example demonstrations provided by a human teacher. While DL approaches allow the robot to learn skills autonomously, they often require programming and domain experts to prepare the system input (*e.g.,* label or preprocess data, define policy and reward functions). Similarly, reinforcement learning generally requires long training times to gather enough data to learn a skill.

In contrast, PbD provides a more intuitive low-effort solution, where the teacher's main task involves providing demonstrations to the robot. Since PbD solutions allow the robot to learn from a sparse set of examples, the data and time required to learn a skill is moderately low. Thus, our work uses PbD to allow end-users to program robots. In the following sections we will give a brief overview of PbD and Automated Planning, the two main concepts used in our proposed framework.

2.1 Programming by Demonstration

PbD, also referred to as *Learning from Demonstration*, is an end-user programming technique for teaching a robot new skills by demonstrating them, without writing code [8]. It has become a central topic in research areas, with the aim to move from purely pre-programmed robots to flexible user-based interfaces for training robots. PbD is traditionally used to learn low-level actions from *trajectory* demonstrations using Gaussian Mixture Models [8,10] or Dynamic Movement Primitives [36]. They can also be learned from *keyframe-based* demonstrations (kfPbD), where the user kinesthetically manipulates the robot's arm to record a series of end-effector poses, referred to as *keyframes* [3]. While demonstrations can be provided in different ways (*e.g.,* by observing a human teacher), users prefer to control the robot directly [46]. In kfPbD, actions are 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. Alexandrova et al. [4] implemented an end-user robot programming system to teach generalisable actions from a single demonstration where keyframes are automatically inferred and actions can be modified retrospectively via a graphical interface.

2.2 Automated Planning

Automated Planning, also known as *AI Planning*, is a research field that focuses on the development of task planners consisting of efficient search algorithms to generate solutions to problems [18]. Given a planning *domain*, *i.e.,* a description of the state of the world and a set of actions, and a planning *problem*, *i.e.,* an initial state (Fig. 1a) and a goal (Fig. 1d), the task planner generates a sequence of actions (Fig. 1c), which guarantees the transition from initial states to the goal states. To allow a correct transition between different world states, high-level actions are defined in terms of preconditions and effects, which represent states before and after the action execution respectively (Fig. 2). 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

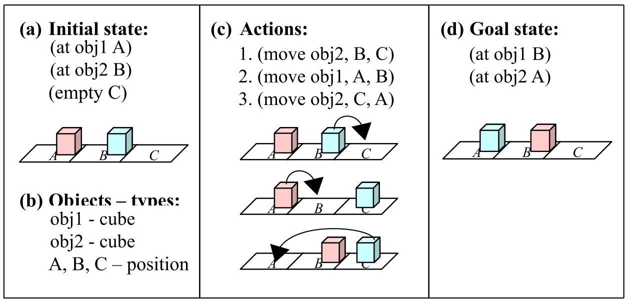


Fig. 1 Definition of a planning problem **a** properties describing the initial world state **b** object names and their types **c** instantiated actions **d** properties describing the goal [27]

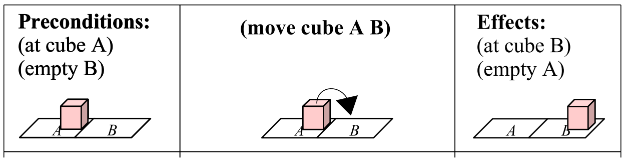


Fig. 2 Action model representation to move a cube from position A to B in terms of preconditions and effects [27]

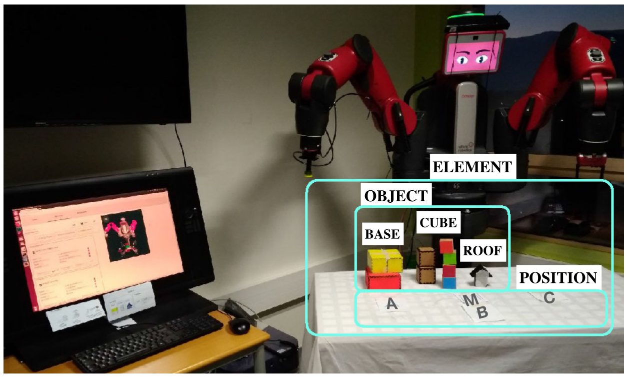


Fig. 3 Experimental setup of the user study. Users programmed the Baxter robot via a graphical interface in order to manipulate objects (shown with their type hierarchies) in the task domain [29]

where $\text{eff}(a) = \text{eff}(a)^- \cup \text{eff}(a)^+$. Action parameters are associated with a *type* (Fig. 1b) and a potential type hierarchy. For example, a type hierarchy, consisting of a general type ELEMENT, can be divided into POSITION and OBJECT, which further divides into BASE, CUBE, and ROOF (Fig. 3). Predicates are defined in first-order logic and used to describe world states and relations between their elements (*e.g.,* `(at obj1 A)`). Planning algorithms use a symbolic planning language as their standard encoding language, such as STRIPS [13] or PDDL [18]. An example of a planning domain in PDDL can be seen in Fig. 4.

The Tower of Hanoi problem could be defined in terms of a planning problem, where the domain consists of 3 pegs and a number of disks. The action is defined as moving a disk from one peg to another, with associated rules as preconditions and effects. A planner can then be used to generate a solution to the problem for any number of disks.

From now on, the word "action" will refer to an action as defined in automated planning. It is worth noting that the predicates used in the planning domains are freely defined by the persons encoding those domains. However, maintaining a standard based on a common vocabulary, with clear and concise definitions is a basic requirement for knowledge representation and reasoning in autonomous robotics to allow the interoperability of robotic systems and communication between robots and humans. The IEEE Standard Association's Robotics and Automation Society recognised this need, and a set of ontologies have been developed [34].

3 Related Work

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 [37,38,45]. For example, Paxton et al. [37] use Behaviour Trees to represent

```
(define (domain iRoPro)
 (:requirements :strips :typing)
 (:types
    element
    position - element
    object - element
    cube - object
    base - object
    roof - object )
 (:predicates
    (clear ?e - element)
    (thin ?o - object)
    (flat ?e - element)
    (on ?o - object ?e - element)
    (stackable ?o - object ?e - element)
 (:action move
  :parameters (?o - object ?A - position ?B - position)
  :precondition (and (on ?o ?A) (clear ?o) (clear ?B)
  :effect (and (on ?o ?B) (clear ?A)
                  not(on ?o ?A) not(clear ?B))
)
```

Fig. 4 Example of a planning domain in PDDL

task plans that are explicitly defined by the user and can be modified to adapt to new tasks. In our work we argue for the use of task planners to automatically generate plans for new scenarios, rather than have the user manually modify them.

Previous work has addressed knowledge engineering tools for constructing planning domains but usually require PDDL experts (PDDL Studio [39]), or common knowledge in software engineering (GIPO [44], itSIMPLE [48]). There has been work on integrating task planning with robotic systems [12,24], learning high-level actions through natural language instructions [43] or learning preconditions and effects of actions to be used in planning [21,23,47]. However, in all of these cases, the robot is provided with a fixed set of low-level motor skills. In our approach, we do not provide the robot with any predefined actions but enable users to teach both low- and high-level actions from scratch.

PbD has been commonly applied to allow end-users to teach robots new actions by demonstration. Alexandrova et al. [4] 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. [1] 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. In this work we argue 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 address previously unseen problems that can be solved with task planners.

4 iRoPro-Interactive Robot Programming

In our previous work [28] we proposed iRoPro, an interactive Robot Programming framework that allows end-users to teach robots new actions that can be reused with task planners. The framework consists of the following three aspects (Fig. 5):

A. Programming by Demonstration: The user *teaches* the robot primitive actions by demonstration. The robot creates an action model that the user can modify and validate.
B. Automated Planning: The user defines a new planning problem with a goal to achieve. The robot *reuses* the taught actions with a planner to generate solutions for new problems.
C. Retro-active Loop: The user observes the robot execution and *refines* taught actions via the graphical interface.

The user is provided with a GUI that abstracts from the underlying modelling language used for Automated Planning. For each step, the user interacts with the GUI to navigate between the components to teach new actions by demonstration, modify inferred action conditions, define new planning problems for the robot to solve and execute generated plans. In the following sections, we give a brief description of each component. We refer the reader to our previous work [27,28] for more details.

4.1 Programming by Demonstration: Teach Actions

Teaching primitive actions consists of learning both *how* and *when* an action should be applied, *i.e.,* learning the low-level action trajectory as in PbD and the high-level representation with preconditions and effects as used in Automated Planning. We consider an action that consists of both low- and high-level representations an *action model*. The high-level representation can be entered directly by the user or inferred from observing the world state before and after the action demonstration. In our work, we first infer the preconditions and effects which the user can subsequently modify on the GUI. We rely on the user's logical reasoning and understanding of what they want to teach the robot and allow them to directly program and correct inferred action models. Thus, the robot can learn a new action from a single demonstration with the user acting as the expert to correct inferred conditions. The user validates the learned action model or provides additional demonstrations to refine the low- or high-level representations. The user repeats this process and creates an action model for each primitive action (see Fig. 5a).

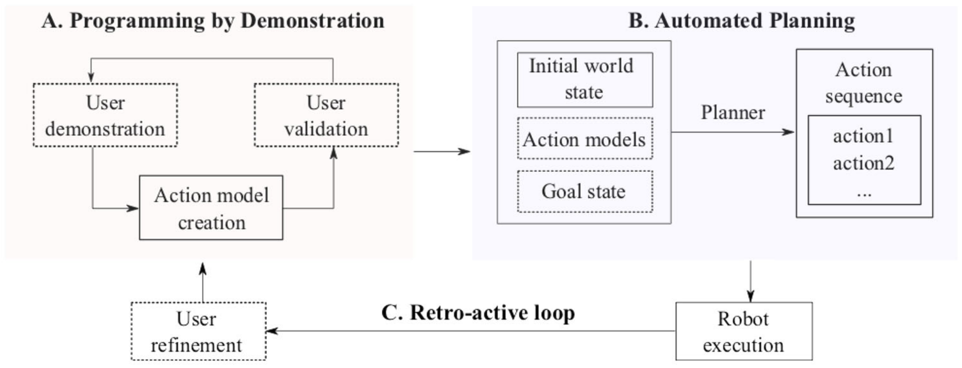


Fig. 5 Overview of the main components of iRoPro: A. the user teaches primitive actions by demonstration B. the robot reuses these with a task planner to generate an action sequence to achieve a goal C. After observing the robot execution, the user can refine the taught action models [27]. Dotted lines indicate user input actions

4.2 Automated Planning: Reuse Actions

In the PbD step, the robot learned action models that include the high-level representation used in Automated Planning. The Automated Planning step consists of creating a planning problem and defining a goal for which the integrated task planner can generate a solution (Fig. 5b). Given a description of a planning *domain*, we can define a planning *problem* with an initial state and a desired goal state to achieve. Depending on the robot architecture and perception system, a partial PDDL domain can be predefined in the system that includes a set of object types and predicates that the robot can recognise. The action models created in the PbD step complete the partial PDDL domain. Similar as before, the robot can infer initial world states for the planning problem and the user can modify and correct them via the GUI. Given the user-defined goal, the task planner generates a plan consisting of an ordered action sequence for the robot to execute. The user can verify the generated plan and have the robot execute it in real life. If no plan is generated or if the plan seems incorrect, the user can modify the taught action models, as well as the initial and goal states and relaunch the planner.

4.3 Retro-Active Loop: Refine Actions

The retro-active loop allows the user to revisit and correct created action models (see Fig. 5c). It is likely that the initially generated plan would not achieve the specified goal, especially if the context of the planning problem is different to that of the initial demonstration (*e.g.,* different object types or positions). Programmed action models can be generalised and reused, especially if the low-level action remains the same. Instead of creating new action models for each new problem, the user can simply modify the action parameters, preconditions or effects. This minimises the user's programming effort and the number of demonstrations required. Thus, the application to a new context is an important step to test the generalisability of action models.

There are several possible causes why the planner might generate incorrect or non-existent solutions:

– *Action parameters* this restricts or generalises the application of the action as they dictate what types an action can be applied to. An action is not considered by the planner, if the types do not match those in the initial state of the planning problem (*e.g.,* pick-and-place was only defined for cube objects but not for other types).
– *Preconditions* similar to action parameters, they define when an action can be applied, but in terms of predicates describing the initial world state. All stated preconditions must hold in a world state in order to apply the action (*e.g.,* pick-and-place of an object only if it is clear).
– *Effects* they define how the world state is updated after the action execution and help the robot to keep track of changes. If they are not defined correctly, there can be a mismatch of the robot's assumed world state and the actual world state (*e.g.,* a position is still considered free when it is occupied).
– *Initial states* they describe the existing world state to the robot. If the initial states are incorrect or missing the planner may consider certain actions as invalid or not find a plan to achieve the goal (*e.g.,* an object is not mentioned in the initial states at all).
– *Goal* the user defines the set of predicates for the robot to achieve. This should not include intermediate steps to achieve the goal nor contradicting states (*e.g.,* 'object is on A' and 'A is clear' are both stated as goal states).

Knowledge engineering tools can facilitate this process of modifying action models. They often provide useful functionalities for dynamic testing, model checking and visualisation [44], but most tools require expertise in Automated Planning or Software Engineering. In our work we argue that the proposed robot programming process does not require this expertise and can be learned easily by users with different educational backgrounds.

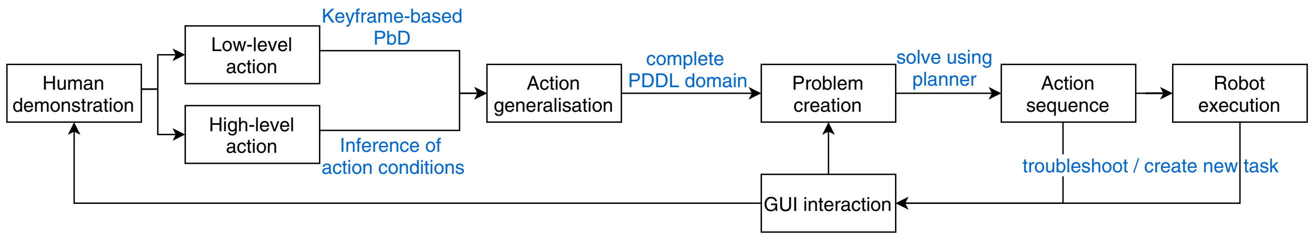


Fig. 6 Overview of iRoPro that allows users to teach low- and high-level actions by demonstration. The user interacts with the GUI to run the demonstration, modify inferred action conditions, create new planning problems for the robot to solve and execute

5 System Implementation

We implemented iRoPro 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 [40] and available online[1] The action is learned by demonstration 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, we define *landmarks* as either predefined table positions or objects that are detected from Kinect point cloud clusters using an open-source tabletop segmentation library[4]. An object $o = (x, y, z, width, length, height)$ is represented by its detected location and bounding box diemensions, which are used to infer its type and related predicates. The user completes the partial PDDL domain via the GUI by creating new action models by demonstration. Then they create planning problems that can be solved with the integrated task planner. Figure 6 shows an overview of the programming process.[5] In the following sections we will give a brief overview of the system implementation. Further details can be found in our previous work [29].

5.1 Low-level Action Representation

We represent low-level actions as proposed in previous work using keyframe-based PbD [4], 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 action demonstration, 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. While these actions can be learned from multiple demonstrations [33], we take the approach that only requires a single demonstration by heuristically assigning poses and letting the user correct them if needed [4]. 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.,* different conditions for actions using claw or suction grippers).

5.2 High-level Action Representation

We implemented a partial PDDL domain with predefined types and predicates that the robot can automatically detect using its sensors. We defined five predicates commonly used for object manipulation tasks and included two (flat and thin) to describe further 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 can be grasped with the claw gripper

In our definition, 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 [26]) but was beyond the scope of this work.

5.3 Action Inference from Demonstration

Instead of manually defining action parameters, preconditions and effects, we accelerate the programming process

[1] https://github.com/ysl208/iRoPro

[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 seen at https://youtu.be/NgaTPG8dZwg

by inferring them from the observed sensor data during the teaching phase. Object types are inferred based on their detected bounding boxes. 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 action demonstration as seen in similar work for learning object manipulation tasks [2]. 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.,*

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

where eff(a) includes positive and negative effects (Fig. 7). A predicate ϕ has variables $\mathrm{var}(\phi) = \{v_1, v_2, \ldots\}$, where each v_i has a type. Therefore, action parameters are the set of variables that appear in either preconditions or effects, *i.e.,*

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

Note that conditions could be learned from multiple demonstrations [1,23]. 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.

5.4 Action Generalisation

The low-level action representation (Sect. 5.1) generalises motion trajectories by re-calculating poses based on detected landmarks from the demonstrated to the new environment. The high-level representation (Sect. 5.2) specifies when an action can be applied, therefore allows taught low-level motion trajectories to be reused for other objects (*e.g.,* use suction gripper for all objects, regardless of their type) or to be restricted for certain types (*e.g.,* only BASE objects). By combining these two representation levels, taught actions can be generalised for new environments that are different to the demonstrated one, allowing the user to customise them for their specific use case.

5.5 Interactive Robot Programming

The user interacts with the GUI that visualises the robot and detected objects (Fig. 8). During the programming process, the GUI allows them to create new actions, run the kinesthetic teaching by demonstration, modify inferred types and

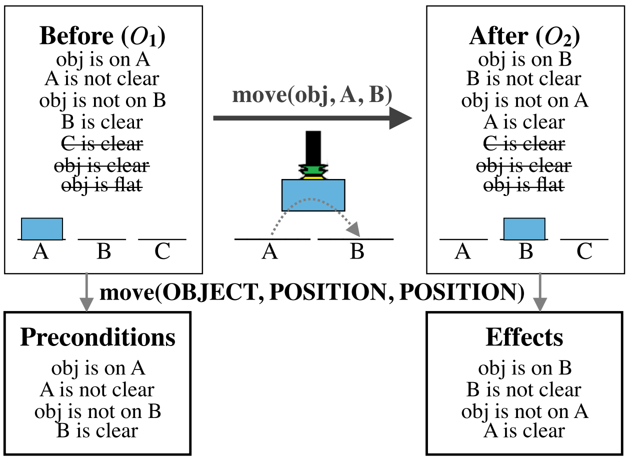
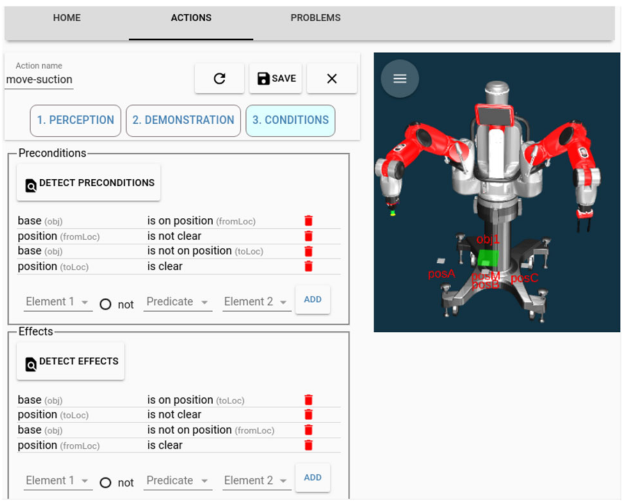


Fig. 7 Example of a high-level action for moving an object from **a** to **b**. Conditions are inferred from the observed predicates before (O_1) and after (O_2) the demonstration [29]



Action name
move-suction

Preconditions

base (obj)	is on position (fromLoc)
position (fromLoc)	is not clear
base (obj)	is not on position (toLoc)
position (toLoc)	is clear

Effects

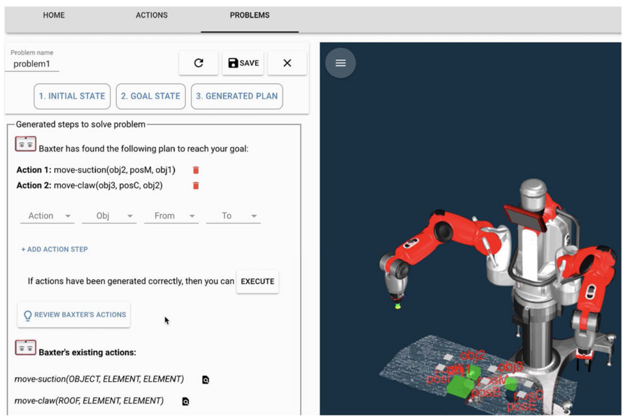
base (obj)	is on position (toLoc)
position (toLoc)	is not clear
base (obj)	is not on position (fromLoc)
position (fromLoc)	is clear

Fig. 8 The iRoPro interface showing the action condition menu and an interactive visualisation of the Baxter robot and detected objects

predicates and to create and solve new problems with the task planner. The interactive robot programming cycle consists of creating and modifying actions and problems:

Actions. New actions are taught by kinesthetically moving the robot's arms using kfPbD [3], where both low-level and high-level actions are learned and generalised (Sect. 5.4). The user can have the robot re-execute the taught action immediately in order to validate it. 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.

Problems. New planning problems can be generated 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



Generated steps to solve problem

Baxter has found the following plan to reach your goal:

Action 1: move-suction(obj2, posM, obj1)

Action 2: move-claw(obj3, posC, obj2)

If actions have been generated correctly, then you can EXECUTE

Baxter's existing actions:

move-suction(OBJECT, ELEMENT, ELEMENT)

move-claw(ROOF, ELEMENT, ELEMENT)

Fig. 9 The iRoPro interface showing the problems menu, the action sequence generated by the task planner and the debug menu to review actions

to achieve. The complete planning domain and problem are translated into PDDL and sent to the Fast-Forward planner [20]. 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 (Fig. 9). If no solution is found or if the generated plan is wrong, the user can open a debug menu to review actions. It provides a summary of the entire planning domain with hints described in natural language to troubleshoot (*e.g.,* 'make sure the action effects can achieve the goal states'). In our post-design study (Sect. 6.4) 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. For any subsequent tasks, the user can create a new problem or modify existing ones by re-detecting the objects.

5.6 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 (Sect. 5.1). To accelerate the execution, we only detect landmarks once at the start, then save their new positions in memory for quick reference, which we refer to as a *mental model*. After each action execution, the mental model is updated with the latest positions of the landmarks according to the action's effects (*e.g.,* obj moved from position A to B). As this assumes that actions are always executed successfully, successful executions need to be checked separately but were beyond the scope of this work. The mental model is also used as a workaround for our limited perception system as it does not detect stacked objects in their initial states (as discussed in Sect. 7).

6 Experimental Evaluation

We first conducted two pre-design experiments to evaluate the usability of the proposed framework. They consisted of qualitative user studies to respond to the following questions:

Q1 How do non-expert users adopt the Automated Planning language with its action model representation? (Sect. 6.1)
Q2 Can users teach a robot action models for Automated Planning using the proposed framework? (Sect. 6.2)

In both experiments we particularly focused on elements to assess the user's understanding of action models such as defining their preconditions and effects. Understanding this symbolic representation is a key requirement to use iRoPro.

At the start of the experiments, the users were introduced to the concept of action models (in terms of preconditions/effects) as being the representation used by the robot. Throughout the experiments, users had to teach new actions and modify their associated conditions. Based on these results, we implemented iRoPro on the Baxter robot (Sect. 5) and subsequently conducted post-design experiments to evaluate the working system with real end-users (Sect. 6.4). Furthermore, we compared results obtained from pre-design with post-design experiments and validate the usability of our proposed framework (Sect. 6.5). In the following sections we give a brief overview of the experimental setup, design, measurements and results. Further details on all experiments can be found in previous work [27,29].

6.1 Acceptance of Automated Planning Concepts

In this experiment, we addressed the following question:

Q1 How do non-expert users adopt the Automated Planning language with its action model representation?

Users ($N = 10$) with little to no programming experience were introduced to the symbolic language and syntax with type structures used in Automated Planning. Users were instructed to describe world state configurations to the robot. The goal was to assess the user's adoption of the planning concepts (*e.g.,* object types, action models) and to verify that the symbolic planning language was appropriate for non-expert users.

6.1.1 Experimental Design and Measurements

The experimental setup consisted of a 2x2 board (with positions A1, A2, B1, B2), 2 cubes, 1 ball, and 1 ball recipient in the form of a bowl. The participants were given sheets with empty tables to complete for each task. Each participant was allocated 1 hour, but the average duration of the

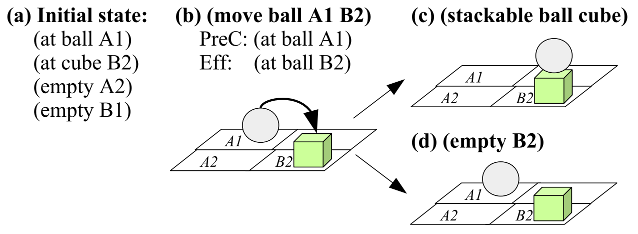


Fig. 10 Users were instructed to provide a description of (**a**) the initial state of the world and **b** an initial move action model. Then they derived additional preconditions for moving the ball from position A1 to B2: (**c**) *(stackable ball cube)*: the ball can be stacked onto the cube, and **d** *(empty B2)*: if the ball cannot be stacked, the target position should be empty [27]

experiment was 49 minutes. Users were told that they needed to use a symbolic planning language to describe the state of the world and the semantic meaning of actions to the robot (Fig. 10). At the course of the experiment, users were faced with three different scenarios of increasing complexity. The participants' behaviour was observed by the experimenter and the experiment was recorded on camera. We evaluated their capability to learn the presented planning language and apply it to different problem statements. At the end, participants were given a questionnaire related to their experience and their understanding of the learned planning language and concepts.

6.2 Acceptance of the Robot Programming Process

In this experiment, we addressed the following question:

Q2 Can users teach a robot action models for Automated Planning using the proposed framework?

Users (N = 11)[6] with different programming experience were presented a simulated implementation of iRoPro and had to teach action models by kinesthetically manipulating a Baxter robot. Users were instructed to teach a primitive action by demonstration and assign preconditions and effects. The goal was to assess the framework's usability and the user's difficulties encountered during the programming process. At the end, participants were given a questionnaire related to their experience, their perceived understanding of the presented concepts and the usability of the framework. In the following sections we briefly outline the experimental setup, measurements and results of the experiment.

The experiments were conducted using a Baxter robot, mounted with a partial implementation of the framework. The implemented functionalities included:

[6] All participants were different from the first experiment.

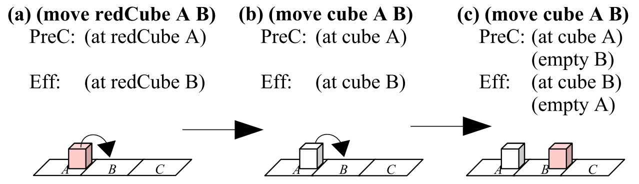


Fig. 11 Continuous refinement of the move action model: **a** initial action model learned by demonstration, **b** action model for all cubes of any colour, **c** action model with an additional condition, if the target position is occupied and cubes can not be stacked

– 'learn new action': record the kinesthetic action demonstration,
– 'find a coloured object': apply the recorded action to an object of the specified colour,
– 'execute an action sequence': execute a sequence of previously taught actions.

We used the Wizard-of-Oz technique to simulate the remaining functionalities (*e.g.,* 'infer action preconditions and effects', 'generate solution using a planner'). Each participant was allocated 1 hour, but the average duration was 29.5 minutes. The participants' behaviour was observed by the experimenter and the experiment was recorded on camera.

6.2.1 Experimental Design and Measurements

The experiment scenario was set in a simulated assembly line, where objects of the same shape, but different colour arrived consecutively at a departure position. Users had to teach the Baxter robot the action for moving an object from the departure position to an arrival position, where another maintenance task would be performed later. Users first demonstrated the action on the robot by guiding its arm through the desired trajectory, then they were presented with an action model that the robot had 'learned'. At the course of the experiment, users were faced with two different scenarios, where they had to suggest logical conditions for the action model in order for the robot to apply them successfully (Fig. 11). We evaluated the user's capability to improve action models and associate conditions when faced with different situations, and assessed the framework's overall usability.

6.3 Pre-Design Experiments Findings

In both experiments, we did not observe a significant difference in the performance between users with different programming experience. In the first experiment, the majority (9 or 90%) of the participants managed to describe the complete world state using the correct syntax. All participants gave correct explanations for preconditions and effects of action models, and provided correct examples. Figure 13a) shows the user responses to the questionnaire in the second experiment. All 11 users were satisfied with the programming process and Baxter's ability to learn and reproduce the demonstrated move action.

The majority of the users had issues formulating the logical properties used for preconditions and effects. In the first experiment (Sect. 6.1), users had difficulties formulating certain conditions in the planning language (*e.g.,* `(stackable ball cube)`), but stated equivalent ones (*e.g., 'only place the ball, if it is stackable on the cube'*). Similarly, in the second experiment (Sect. 6.2), users formulated missing preconditions (*e.g., 'position B is empty'*) with other equivalent conditions (*e.g., 'do not place the object on position B, if it is occupied'*).

Some of the users made wide assumptions about the robot's capabilities. In the second experiment, when both arrival and departure positions were occupied, 5 (or 50%) of the users expected Baxter to consider the occupied position, even though the condition was not mentioned in its action model. This is a common problem in PbD solutions as there is a difference between the robot's intelligence and the one perceived by its teacher [46]. This can be addressed by reproducing the learned action in a new context and verifying the robot's knowledge base, as we did throughout the experiment.

With these two qualitative experiments, we showed that the Automated Planning language and its main concepts can easily be learned by users without any programming background. The action model representation, in terms of preconditions and effects, seems to be intuitive for non-expert users. These initial experiments provided us with an initial idea of how the users might perceive the proposed robot programming framework. We intentionally limited the set of Automated Planning concepts (*i.e.,* object types, predicates, and actions with preconditions and effects) that are necessary to use the framework to the bare minimum to assess the potential usability of such a framework. Further experiments should test scalability, address more Automated Planning concepts (*e.g.,* object-type hierarchy, more predicates, planning problem definition and resolution) and potentially compare separate control groups (*e.g.,* experts vs non-experts) in less structured environments.

6.4 Post-Design Experiment

The second part of the evaluation was conducted using THEDRE [32], 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 (Sect. 5.3) 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

In the following we will give a brief overview of the experiment setup, design, measurements and results. Further details can be found in our previous work [29].

6.4.1 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.

6.4.2 Protocol

Users were first given a brief introduction to task planning concepts, the Baxter robot and the experimental set up (Fig. 3). 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 (Table 1), where the first two were practice tasks to introduce them to the system. 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 (Fig. 12). Otherwise, the user had to modify the existing input until the plan was

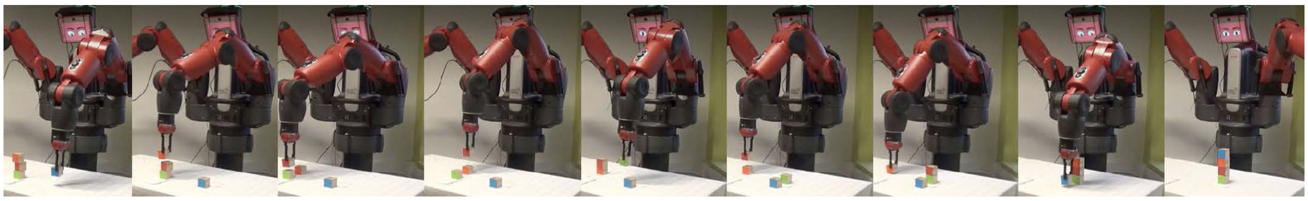


Fig. 12 Snapshots of an example task execution for stacking CUBE objects with the claw gripper

Table 1 Benchmark tasks for the user study where the first two tasks were used to introduce participants to the system [29]

# 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

correctly generated. Tasks 6-8 were similar to the previous tasks (1-5) but use both robot grippers.

6.4.3 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 seven 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) [9] 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 compared overall responses to the second user study, where users were simulated a similar robot programming experience using the Wizard-of-Oz technique but had no direct interaction with a working system (Sect. 6.2). Finally, participants were asked which aspects they found most useful, most difficult, and which they liked the best and the least.

6.5 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 2):

(H1)–(H3) User performance

Users took between 22 and 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 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.

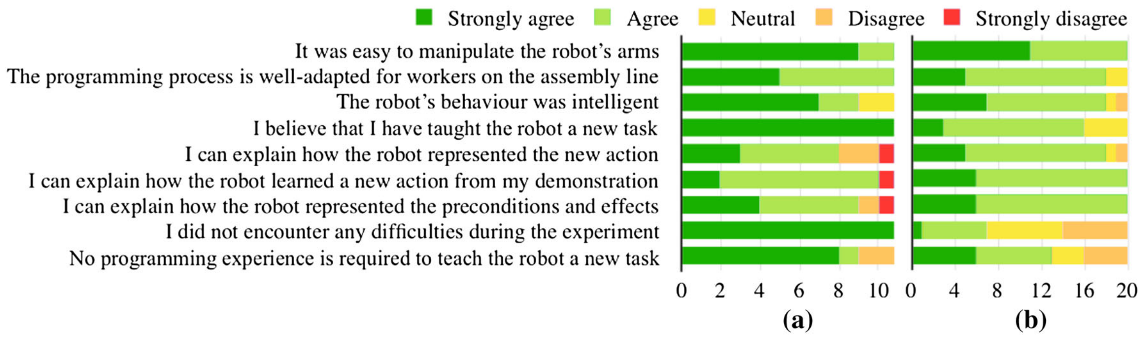


Fig. 13 User responses from the post-study survey comparing (**a**) the pre-design study (Sect. 6.2) with (**b**) the post-design study (Sect. 6.4)

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 [9] obtained from the post-study survey. Using Bangor et al. 's categories [6], 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 [41], 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.

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%).

Table 2 User performance comparing task completion times with pre-study test scores [29]

	Main tasks (in min)		Pre-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

6.6 Pre- versus Post-Design Experiments

We compare post-study questionnaire responses between pre- and post-design experiments (Fig. 13). In the first experiment (Sect. 6.2), users had no direct interaction with the robot programming system as it was simulated using the Wizard-of-Oz technique, while in the last experiment (Sect. 6.4), users programmed the robot using the end-to-end system implementation. The main differences were noted regarding difficulties encountered during the experiment: In the first study 11 (or 100%) users agreed that they encountered no difficulties, while in the last study only 7 (or 35%) users stated the same. However, all users in the last study 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 the first study. Both differences can be explained by the fact that in the last study, users had to use an end-to-end system to program the robot, while in the first study, users had no direct interaction with a working system. Even though users encountered more difficulties in the last study, 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, both user studies received positive responses. While the taught concepts seem quite intuitive to most users, some of them faced problems throughout the experiment. With these questions, we wanted to measure how well they perceived their own understanding of the robot programming process.

6.7 Continuous Improvement of the System

The system underwent four phases of improvement, allowing us to refine the system functionalities, GUI and user instruction methods:

1. *Initial prototype* Based on the feedback received on our initial prototype by domain experts, we changed the flow for introducing the system to novice users as it included a lot of new information (*e.g.,* PbD and Automated planning concepts) that they were not familiar with.
2. *Pre-tests with 3 users* We ran pre-tests with 3 users who have never seen the system before and further improved the experiment flow (*e.g.,* create an action for one object at a time). We also made the user interface more friendly to include Baxter icons (Fig. 14) and eyes that followed the robot's moving joint [17]) on the robot's screen so that it seemed more human-like. We noticed that troubleshooting incorrect actions seemed difficult for all users, so we included an option to review actions and goals which provides a summary of the created input and hints to guide the debugging steps.
3. *First experimental tests with 5 users with condition inference* After running the experiments with 5 users, we noticed that the majority did not modify the inferred action conditions. This raised an interesting HRI question of whether users who were given automatically generated conditions would 'blindly trust' them. Hence, we decided to create two experiment groups: with and without condition inference, where the first 5 participants belonged to the former. Participants in the latter group would need to manually enter all conditions via the interface.

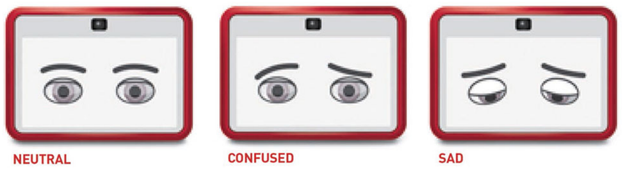


Fig. 14 Baxter icons used for the graphical interface [15]

4. *Final experimental tests with 16 users* The remaining users were divided into the two control groups so that we had an equal number of participants in both control groups, while also maintaining an even distribution of programming levels. The results of the 21 participants in the experimental tests were used as the final evaluation of the system as discussed in the previous sections.

7 Discussion

We demonstrated that iRoPro can be used to generalise primitive actions to a range of complex manipulation tasks and that it is easy to learn for users with or without programming experience. In our system evaluation we could have programmed other actions, such as turning or pushing for packaging tasks [30]. As the purpose of our evaluation was to show the generalisability of primitive actions with the use of a task planner, we decided to stick to pick-and-place actions with two different grippers. Limitations and interesting extensions of our work are as follows:

1. Our object perception is limited as it does not detect objects that are too close together (*e.g.,* stacked objects). An improved perception system would allow the detection of initial states with stacked objects, automatically detecting goal states, or verifying action executions.
2. Due to the different grippers, we did not program actions that use both arms (*e.g.,* carrying a tray). A possible extension would be to include a better motion and task planning system in order to allow executing both arms simultaneously while avoiding self-collision.
3. We only included a minimal set of predicates (Sect. 5.2) that we deemed intuitive and useful for object manipulation tasks. It could be interesting to include and learn predicates to capture more complex domains such as object orientation [26].
4. iRoPro can easily adapted to different kinds of robots: the source code is available online (Sect. 5) and developed in ROS, which is a popular framework that facilitates the interoperability of a wide variety of robotic platforms. PDDL is platform-independent and a standard used to benchmark task planners. Moreover, bridging PDDL and

IEEE Standard Ontologies for Robotics and Automation has been studied previously [16,34].

8 Conclusion

We presented iRoPro, an interactive Robot Programming system that allows simultaneous teaching of low- and high-level actions by demonstration. The robot reuses the actions with a task planner to generate solutions to tasks that go beyond the demonstrated action. We conducted pre-design experiments to evaluate the feasibility of the proposed framework and involved concepts in PbD and Automated Planning. Then, we implemented the system on a Baxter robot and showed its generalisability on six benchmark tasks by teaching a minimal set of primitive actions that were reused for all tasks. We demonstrated its usability with a user study where participants with diverse educational backgrounds and programming levels learned how to use the system in less than an hour. Furthermore, we compared user responses from our pre-design experiments with our post-design evaluation and investigate discrepancies. 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 tasks. Thus, iRoPro enables end-users to program robots from scratch, without writing code, therefore maximising the generalisability of taught actions with minimum programming effort.

9 Future Work

Future work will focus on exploring more challenging domains to extend the system to other platforms by including a wider range of predicates and probabilistic techniques to improve the condition inference. The next step is to focus on less controlled environments involving factory workers who may ultimately use this technology as well as mobile robots that move around between work spaces. This will require more complex planning domains to consider planning between robots and humans, human-robot collaborative tasks as well as solutions for multi-modal communication involving natural cues [35].

Acknowledgements The authors would like to thank Nadine Mandran, research engineer at the Laboratoire d'Informatique de Grenoble, for her support and guidance during for the experimental studies.

References

1. Abdo N, Kretzschmar H, Spinello L, Stachniss C (2013) Learning manipulation actions from a few demonstrations. In: International conference on robotics and automation, pp 1268–1275. IEEE
2. Ahmadzadeh SR, Paikan A, Mastrogiovanni F, Natale L, Kormushev P, Caldwell DG (2015) Learning symbolic representations of actions from human demonstrations. Int Conf Robot Autom 5:3801–3808
3. Akgun B, Cakmak M, Jiang K, Thomaz AL (2012) Keyframe-based learning from demonstration. Int J Social Robot 4(4):343–355
4. Alexandrova S, Cakmak M, Hsiao K, Takayama L (2014) Robot programming by demonstration with interactive action visualizations. Robot Sci Syst 3:10
5. Argall BD, Chernova S, Veloso M, Browning B (2009) A survey of robot learning from demonstration. Robot Auton Syst 57(5):469–483
6. Bangor A, Kortum PT, Miller JT (2008) An empirical evaluation of the system usability scale. Int J Hum Comput Interact 24(6):574–594
7. Biggs G, Macdonald B (2003) A survey of robot programming systems. Austral Conf Robot Autom 3:1–3
8. Billard A, Calinon S, Dillmann R, Schaal S (2008) Robot programming by demonstration. In: Springer handbook of robotics, pp 1371–1394. Springer
9. Brooke J (2013) SUS: a retrospective. J Usability Stud 8(2):29–40
10. Calinon S, Billard A (2007) Incremental learning of gestures by imitation in a humanoid robot. In: International conference on human-robot interaction, pp 255–262
11. Calzado J, Lindsay A, Chen C, Samuels G, Olszewska JI (2018) Sami: interactive, multi-sense robot architecture. In: 2018 IEEE 22nd international conference on intelligent engineering systems (INES), pp 317–322
12. Cashmore M, Fox M (2015) ROSPlan: planning in the robot operating system. In: International conference on automated planning and scheduling, pp 333–341
13. Fikes RE, Nilsson NJ (1971) STRIPS: a new approach to the application of theorem proving to problem solving. Artif Intell 2(3–4):189–208
14. Fraser N (2013) Blockly: a visual programming editor. https://developers.google.com/blockly/. Accessed 01 April 2019
15. Freedman DH (2012) The Rise of the Robotic Work Force. https://www.inc.com/magazine/201210/david-h-freedman/the-rise-of-the-robotic-workforce.html. Accessed 11 April 2019
16. Pignaton de Freitas E, Bermejo-Alonso J, Khamis A, Li H, Olszewska JI (2020) Ontologies for cloud robotics. Knowl Eng Rev 35:e25. https://doi.org/10.1017/S0269888920000338
17. Gaisne A (2018) Baxter eyes. https://github.com/Anne-Gaisne/baxter_eyes. Accessed 11 April 2019
18. Ghallab M, Nau D, Traverso P (2004) Automated planning: theory and practice. Elsevier, London
19. Gosavi A (2009) Reinforcement learning: a tutorial survey and recent advances. J Comput 21(2):178–192
20. Hoffmann J, Nebel B (2001) The FF planning system: fast plan generation through heuristic search. J Artif Intell Res 14:253–302
21. Jetchev N, Lang T, Toussaint M (2013) Learning grounded relational symbols from continuous data for abstract reasoning
22. Kaelbling LP, Littman ML, Moore AW (1996) Reinforcement learning: a survey. J Artif Intell Res 4:237–285
23. Konidaris G, Kaelbling LP, Lozano-Perez T (2018) From skills to symbols: learning symbolic representations for abstract high-level planning. J Artif Intell Res 61:215–289
24. Kuhner D, Aldinger J, Burget F, Göbelbecker M, Burgard W, Nebel B (2018) Closed-loop robot task planning based on refer-

ring expressions. In: International conference on intelligent robots and systems, pp 876–881. IEEE
25. Lego: Lego mindstorms. http://mindstorms.lego.com (2003). Accessed 12 March 2019
26. Li C, Berenson D (2016) Learning object orientation constraints and guiding constraints for narrow passages from one demonstration. In: International symposium on experimental robotics, pp 197–210. Springer
27. Liang YS, Pellier D, Fiorino H, Pesty S (2017) Evaluation of a robot programming framework for non-experts using symbolic planning representations. In: International symposium on robot and human interactive communication
28. Liang YS, Pellier D, Fiorino H, Pesty S (2017) A framework for robot programming in cobotic environments: First user experiments. In: International conference on mechatronics and robotics engineering, pp 30–35. ACM
29. Liang YS, Pellier D, Fiorino H, Pesty S (2019) End-user programming of low- and high-level actions for robotic task planning. In: International symposium on robot and human interactive communication
30. Liang YS, Pellier D, Fiorino H, Pesty S, Cakmak M (2018) Simultaneous end-user programming of goals and actions for robotic shelf organization. In: 2018 international conference on intelligent robots and systems
31. Majed M (2014) Learn to program with Scratch: a visual introduction to programming with games, art, science, and math. No Starch Press Inc
32. Mandran N, Dupuy-Chessa S (2017) THEDRE: a traceable process for high quality in human centred computer science research. In: International conference of System Development
33. Niekum S, Osentoski S, Konidaris G, Barto AG (2012) Learning and generalization of complex tasks from unstructured demonstrations. In: International conference on intelligent robots and systems, pp 5239–5246. IEEE
34. Olszewska JI, Barreto M, Bermejo-Alonso J, Carbonera J, Chibani A, Fiorini S, Goncalves P, Habib M, Khamis A, Olivares A, de Freitas EP, Prestes E, Ragavan SV, Redfield S, Sanz R, Spencer B, Li H (2017) Ontology for autonomous robotics. In: 2017 26th IEEE international symposium on robot and human interactive communication (RO-MAN), pp 189–194
35. Pais AL, Argall BD, Billard AG (2013) Assessing interaction dynamics in the context of robot programming by demonstration. Int J Social Robot 5(4):477–490
36. Pastor P, Hoffmann H, Asfour T, Schaal S (2009) Learning and generalization of motor skills by learning from demonstration. In: International conference on robotics and automation, pp 763–768
37. Paxton C, Hundt A, Jonathan F, Guerin K, Hager GD (2017) CoSTAR: instructing collaborative robots with behavior trees and vision. In: International conference on robotics and automation, pp 564–571
38. Perzylo A, Somani N, Profanter S, Kessler I, Rickert M, Knoll A (2016) Intuitive instruction of industrial robots: semantic process descriptions for small lot production. In: International conference on intelligent robots and systems, pp 2293–2300
39. Plch T, Chomut M, Brom C, Barták R (2012) Inspect, edit and debug PDDL documents: simply and efficiently with PDDL studio. In: International conference on automated planning and scheduling: system demonstration
40. Quigley M, Faust J, Foote T, Leibs J (2009) ROS: an open-source Robot Operating System. In: ICRA workshop on open source software
41. Sauro J (2012) Predicting net promoter scores from system usability scale scores. https://measuringu.com/nps-sus/. Accessed 16 April 2019
42. Schmidhuber J (2015) Deep learning in neural networks: an overview. Neural Netw 61:85–117
43. She L, Cheng Y, Chai JY, Jia Y, Yang S, Xi N (2014) Teaching robots new actions through natural language instructions. In: International symposium on robot and human interactive communication, pp 868–873
44. Simpson RM, Kitchin DE, McCluskey TL (2007) Planning domain definition using GIPO. Knowl Eng Rev 22(2):117–134
45. Stenmark M, Haage M, Topp EA (2017) Simplified programming of re-usable skills on a safe industrial robot: Prototype and evaluation. In: International conference on human-robot interaction, pp 463–472. ACM
46. Suay HB, Toris R, Chernova S (2012) A practical comparison of three robot learning from demonstration algorithm. Int J Social Robot 4(4):319–330
47. Ugur E, Piater J (2015) Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning. In: International conference on robotics and automation, pp 2627–2633
48. Vaquero TS, Silva JR, Tonidandel F, Beck JC (2013) itSIMPLE: towards an integrated design system for real planning applications. Knowl Eng Rev 28(2):215–230

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.