



End-user Robot Programming in Cobotic Environments

Ying Siu Liang

► To cite this version:

Ying Siu Liang. End-user Robot Programming in Cobotic Environments. Robotics [cs.RO]. Université Grenoble Alpes, 2019. English. NNT : 2019GREAM020 . tel-02275084

HAL Id: tel-02275084

<https://theses.hal.science/tel-02275084>

Submitted on 30 Aug 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

Spécialité : Informatique

Arrêté ministériel : 25 mai 2016

Présentée par

Ying Siu LIANG

Thèse dirigée par **Sylvie PESTY**, Professeur, UGA
et codirigée par **Damien PELLIER**, MCF, UGA
préparée au sein du **Laboratoire d'Informatique de Grenoble**
dans l'**École Doctorale Mathématiques, Sciences et
technologies de l'information, Informatique**

Programmation d'un robot par des non- experts

End-user Robot Programming in Cobot Environments

Thèse soutenue publiquement le **12 juin 2019**,
devant le jury composé de :

Madame SYLVIE PESTY

PROFESSEUR, UNIVERSITE GRENOBLE ALPES, Directeur de thèse

Monsieur DAMIEN PELLIER

MAITRE DE CONFERENCES, UNIVERSITE GRENOBLE ALPES, Co-
directeur de thèse

Monsieur RACHID ALAMI

DIRECTEUR DE RECHERCHE, CNRS DELEGATION OCCITANIE
OUEST, Rapporteur

Monsieur OLIVIER LY

PROFESSEUR, UNIVERSITE DE BORDEAUX, Rapporteur

Monsieur HUMBERT FIORINO

MAITRE DE CONFERENCES, UNIVERSITE GRENOBLE ALPES,
Examineur

Monsieur DOMINIQUE DUHAUT

PROFESSEUR, UNIVERSITE BRETAGNE-SUD, Président



Abstract

The increasing presence of robots in industries has not gone unnoticed. Cobots (collaborative robots) are revolutionising industries by allowing robots to work in close collaboration with humans. Large industrial players have incorporated them into their production lines, but smaller companies hesitate due to high initial costs and the lack of programming expertise. Recent work has focused on enabling end-users to program robots but teaching reusable actions from scratch is generally left up to robotic experts. In this thesis, we propose a framework that combines two disciplines, Programming by Demonstration and Automated Planning, to enable users with little to no technical background to program a robot. The user constructs the robot's knowledge base by teaching it new actions by demonstration, and associates their semantic meaning via a graphical interface to enable the robot to reason about them. The robot adopts a goal-oriented behaviour by using automated planning techniques to reuse taught actions to generate solutions for previously unseen tasks. We first present preliminary work in terms of user experiments using a Baxter Research Robot to evaluate the feasibility of our approach. We conducted qualitative user experiments to evaluate the user's understanding of the symbolic planning language and the usability of the proposed framework's programming process. We showed that users with little to no programming experience can adopt the symbolic planning language and understand the proposed programming process. We then present a goal-oriented programming system for robotic shelf organisation tasks that uses Programming by Demonstration to simultaneously teach goals and actions. Based on the obtained results and the developed system, we present an implementation of iRoPro, a working end-to-end system that allows teaching low- and high-level actions by demonstration to be reused with a task planner. We evaluate the generalisation power of the system and show how taught actions can be reused for more complex tasks. Finally, we validated the system's usability with a user study and demonstrated that users with any programming level and educational background can easily learn and use the proposed robot programming system.

Résumé

La robotique est de plus en plus présente dans l'industrie. Les cobots (robots collaboratifs), travaillant en collaboration avec l'homme, sont en train de révolutionner l'industrie. Les grands acteurs industriels les ont déjà intégrés dans leurs chaînes de production, mais les petites entreprises hésitent encore en raison des coûts élevés des investissements nécessaires et de leur manque d'expertise dans leur maintenance et leur programmation. Des travaux récents se sont concentrés sur la possibilité, pour des utilisateurs finaux, de programmer des cobots, mais la programmation des actions qu'ils peuvent réaliser reste encore un problème d'expert en robotique. Dans cette thèse, nous proposons un système qui combine deux disciplines de l'IA, l'apprentissage par démonstration et la planification automatique, pour permettre à des utilisateurs ayant peu ou pas de connaissances en programmation. Dans un premier temps, l'utilisateur apprend de nouvelles actions au cobot par démonstration et leur associe une sémantique via une interface graphique. Dans un second temps, le cobot est capable en utilisant des algorithmes de planification automatique de raisonner sur la sémantique des actions apprises de calculer dynamiquement la séquence d'actions à réaliser pour atteindre un objectif qui lui a été confié. Nous présentons dans ce manuscrit tout d'abord des travaux préliminaires pour évaluer la faisabilité de notre approche. Ces travaux préliminaires s'appuient sur une expérimentation qualitative réalisée auprès d'utilisateurs afin d'évaluer leur compréhension du langage de planification symbolique utilisé pour définir la sémantique des actions apprises au cobot. Nous avons montré que les utilisateurs ayant peu ou pas d'expérience en programmation peuvent adopter ce langage et comprendre le processus de programmation proposé. Nous présentons ensuite une méthode pour apprendre au cobot par démonstration un objectif à réaliser. Nous avons évalué notre approche sur une tâche de packaging classique en robotique. Finalement, sur la base des résultats obtenus, nous présentons une implémentation de notre système de programmation par démonstration, appelé iRoPro, un système fonctionnel complet qui permet d'apprendre à un cobot des actions de bas niveau (trajectoire) et de haut niveau (sémantique) par démonstration capable d'être manipulée par un planificateur de tâches. Notre système a été évalué et validé par une étude d'utilisateurs. Nous avons montré que les utilisateurs avec des connaissances en programmation limitées pouvaient relativement facilement apprendre et utiliser notre système de programmation par démonstration.

Acknowledgements

First of all, I would like to thank my parents P.Y. and H.K. Liang for the love, guidance and support they have shown me in every step of my life. I would also like to thank my brother C.T. and other family members for being there for me when I needed.

Thanks to my thesis advisors Damien Pellier, Humbert Fiorino, and Sylvie Pesty for their supervision and guidance, as well as colleagues and previous members or interns in the HAWAI (formerly MAGMA) team for their support throughout my thesis. My sincere gratitude also goes to Nadine Mandran, who was always happy to help me with my user experiments and provided me with essential feedback on both the design and the write-up. Similarly, I would like to thank other members of the lab that I worked with and that helped me with any aspects of my thesis.

I am also grateful to my doctoral school advisor Pierre Genevès and my external advisor Dominique Duhaut for their support. I still remember the time when I just finished my Master thesis presentation in 2016 and after my presentation Dominique told me “Someone like you should do a Ph.D.” and now I did - thanks for believing in me. Also a big thank you to my jury members or *rapporteurs* Rachid Alami and Olivier Ly for reading my thesis, as well as your time and constructive feedback.

Finally, I would like to express my sincere gratitude to Maya Cakmak and the Human-Centered Robotics Lab, for inviting me to work with her for 5 months of my Ph.D. It was truly one of the most fun and inspiring experiences for me to continue work in robotics research after my thesis. I would also like to thank my friends in Grenoble, Seattle, London, and anywhere else in the world who supported me throughout my thesis.

Contents

1	Introduction	14
1.1	Context	14
1.1.1	Cobotics	14
1.1.2	Robot Programming by Demonstration	15
1.1.3	Automated Planning	17
1.2	Problem Statement	17
1.3	Summary of contributions	19
1.4	Document organization	20
I	Literature Review	22
2	Robot Programming	23
2.1	Introduction	23
2.2	Manual Programming Systems	25
2.2.1	Text-based Systems	27
2.2.2	Graphical Systems	27
2.3	Automatic Programming Systems	27
2.3.1	Policy derivation techniques	29
2.3.2	Learning from data	31
2.3.3	Machine Learning Systems	31
2.3.4	Programming by Demonstration	32
2.4	End-user Involvement	34
2.5	Discussions and Relations to Present Work	35
3	Automated Planning	39
3.1	Introduction	39
3.2	Classical planning	40
3.3	Planning Domain Definition Language (PDDL)	42
3.3.1	Planning domain description	43

3.3.2	Planning problem description	45
3.3.3	PDDL evolution and extensions	46
3.4	Planners	47
3.5	Knowledge Engineering Tools	49
3.6	Discussions and Relations to Present Work	49
II	Contributions	53
4	An End-User Robot Programming Framework for Cobotic Environments	55
4.1	iRoPro - interactive Robot Programming	56
4.1.1	Programming by Demonstration: teaching actions	57
4.1.2	Automated Planning: reusing actions	58
4.1.3	Retro-active Loop: refining actions	58
4.2	Methodology	59
5	Pre-Experiments	61
5.1	Baxter Research Robot	62
5.2	Experiment 1: Acceptance of Automated Planning and PDDL Concepts	62
5.2.1	Experimental Setup & Participants	63
5.2.2	Experimental Design & Measurements	64
5.2.3	Results	66
5.3	Experiment 2: Acceptance of the Robot Programming Framework	67
5.3.1	Experimental Setup & Participants	67
5.3.2	Experimental Design & Measurements	68
5.3.3	Results	69
5.4	Findings	70
5.5	Conclusions	71
6	Goal-Oriented End-User Robot Programming	73
6.1	Use Case	74
6.1.1	Related work	76
6.2	Approach	77
6.2.1	Freiburg Dataset Analysis	77
6.2.2	Shelf Arrangement Representation	79
6.2.3	Goal Inference from Demonstration	80
6.2.4	Goal Inference with Direct Specification	81
6.2.5	Action Representation and Inference	81
6.2.6	Implementation	82

6.3	Goal Inference Evaluation	82
6.3.1	Teaching Strategies	83
6.3.2	Evaluation on the Freiburg Dataset	84
6.3.3	User Study Evaluation	86
6.4	System Evaluation	89
6.4.1	Protocol	90
6.4.2	Results	90
6.5	Discussions	90
6.6	Conclusions	92
7	End-User Robot Programming for Task Planning	93
7.1	Related Work	95
7.2	Approach	96
7.2.1	Low-level Action Representation	96
7.2.2	High-level Action Representation	97
7.2.3	Action Inference from Demonstration	97
7.2.4	Action Generalisation	99
7.3	System Implementation	99
7.3.1	Interactive Robot Programming	99
7.3.2	Plan Execution	101
7.4	System Evaluation	101
7.4.1	Protocol	102
7.4.2	Results	102
7.5	User Evaluation	103
7.5.1	Experimental Setup & Participants	103
7.5.2	Experimental Design & Measurements	104
7.5.3	Results	105
7.5.4	Continuous Improvement of the System	108
7.6	Discussions	110
7.7	Conclusion	110
8	Conclusion and Future Work	111
8.1	Contributions	112
8.1.1	End-user robot programming framework	112
8.1.2	Experimental findings	112
8.1.3	Goal-oriented robot programming	113
8.1.4	iRoPro - System implementation	113
8.1.5	User and system evaluation	113
8.2	Limitations	114

8.3 Future work	115
References	117
A Publications and Submissions	132
B Resources for Pre-Experiment: Study 1 (Sec. 5.2)	133
C Resources for Pre-Experiment: Study 2 (Sec. 5.3)	151
D Resources for User Evaluation (Chapter 6)	163
E Resources for User Evaluation (Sec. 7.5)	164
F PDDL code used in Chapter 7	182
F.1 iRoPro planning domain	182
F.2 iRoPro planning problems	184

List of Figures

1.1	Overview of the Programming by Demonstration life-cycle consisting of three steps	16
1.2	Tower of Hanoi problem where the goal is to move the entire stack of disks from one peg to another	16
1.3	Illustration of the main Automated Planning concepts	17
1.4	Overview of thesis chapters	21
2.1	Classical robot programming process	24
2.2	Overview of Robot Programming approaches based on control over robot behaviour code, type of learning data and how it is acquired and teacher feedback. Approaches are ranked by end-user involvement in the programming process as well as data and time required to learn a skill.	26
2.3	KUKA Robotics graphical user interface (Abdeetdal [2017])	28
2.4	Lego Mindstorms EV3 icon-based interface (Lego [2003])	29
2.5	Scratch: block-based visual programming language (Majed [2014])	29
2.6	Sample demonstrations of the letter P in 2D: (a) trajectory demonstration (b) keyframe-based demonstration (c) hybrid demonstration (Akgun et al. [2012]).	30
2.7	Robot programming by keyframe-based demonstration using a graphical user interface that allows retrospectively editing (Alexandrova et al. [2014])	30
2.8	Types of Machine learning: Deep learning (supervised and unsupervised learning) and Reinforcement learning (Jones [2017])	32
3.1	A state transition system consisting of six states s_0 to s_5	40
3.2	Example of a type hierarchy showing a general type <i>ELEMENT</i> with subtypes <i>OBJECT</i> and <i>POSITION</i> , and <i>BASE</i> , <i>CUBE</i> , <i>ROOF</i> objects.	41
3.3	Action model representation of a move action in terms of preconditions and effects: an action (or instantiated operator) for a cube (top), and generalised action (or planning operator) for any object, where variables are prefixed with ‘?’ (bottom).	45

3.4	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 state	46
3.5	Screenshot of a Planning UML model in itSIMPLE software (Vaquero et al. [2013])	51
3.6	Screenshot of the operator editor tool in GIPO (McCluskey and Simpson [2005])	51
4.1	An overview of the iRoPro (interactive Robot Programming) framework: 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 (dotted lines indicate user actions, solid lines indicate robot actions).	56
4.2	The thesis contributions align with the design wheel consisting of successive cycles of ‘Explore’, ‘Create’, ‘Evaluate’ and ‘Manage’ phases (University of Cambridge [2013]).	60
5.1	Experimental setup for user studies.	63
5.2	Action model representation of a move action in terms of preconditions and effects: an action (or instantiated operator) for a cube (top), and generalised action (or planning operator) for any object, where variables are prefixed with ‘?’ (bottom).	64
5.3	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) (<i>stackable ball cube</i>): the ball can be stacked onto the cube, and (d) (<i>empty B2</i>): if the ball cannot be stacked, the target position should be empty.	65
5.4	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.	65
5.5	Extract of questionnaire responses from Experiment 1.	66
5.6	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.	69
5.7	Summary of questionnaire responses: Extract of 18 questions on the user’s perceived usability and understanding of the programming process after the experiment.	70

6.1	Overview of the developed system that allows users to demonstrate part of a shelf arrangement task and interact with a GUI to simultaneously program both the complete task goal (fully specified shelf arrangement) and the actions for achieving that goal.	75
6.2	Main product shape categories, that comprise 90.1% of items in the dataset, are defined based on the the shape of their base and their tip. The shape of the tip determines whether the objects can be stacked or not. The shape of the base determines whether the object is more compactly arranged on a regular grid (rectangular) or an off-grid arrangement (circular).	78
6.3	Cylindrical item arranged in a) regular grid and b) off-grid configurations. c) Soft-packaged item arranged in interleaved grid configuration.	79
6.4	The extended Rapid PbD interface where users can directly specify their desired object arrangement (left) and the inferred arrangement is visualised (purple) overlaying the detected objects (green).	83
6.5	Learning curve of 10 teaching strategies for 4 different shelf arrangements from the Freiburg dataset, with number of actions (x-axis) and ranking of the ground truth shelf arrangement (y-axis).	85
6.6	Average number of steps required to infer the correct arrangement per teaching strategy across the dataset (error bars show standard deviation).	85
6.7	Simplified user-interface created to evaluate our goal inference model in an online user study. The top part shows the desired configuration with a picture and the visualisation of the current shelf arrangement inference. The bottom part has three parts corresponding to three types of programming actions the user can take: demonstration (drag-and-drop items onto shelf), specification (select parameter values from drop-down menus), and selection (choose one of the top four most likely arrangements by clicking on it).	87
6.8	Average number of steps required for 8 arrangement tasks chosen for the online user study, comparing 10 teaching strategies with human performance (AMT).	88
6.9	Distribution of user strategies employed in the AMT study	89
6.10	Snapshots from the executions of the eight system evaluation benchmark tasks.	91
7.1	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.	94
7.2	Experimental setup for the user study (N=21), where users programmed the Baxter robot via a GUI to manipulate given object types (with predefined type hierarchy).	96

7.3	Example of a high-level action for moving an object from A to B. Conditions are inferred from the observed predicates before (O_b) and after (O_a) the demonstration.	98
7.4	The iRoPro interface showing the action condition menu and an interactive visualisation of the Baxter robot and detected objects.	100
7.5	Snapshots from the executions of the system evaluation (Tasks 3&4) showing a claw grip from the top and the side.	103
7.6	Participants who did better in the pre-test questionnaire completed the main tasks faster, with ‘non-CS’ users scoring the highest and being the fastest on average.	107
7.7	User responses from the post-study questionnaires comparing a) the iRoPro user evaluation (N=20) with responses obtained in b) the pre-experiment user study (N=11).	108
7.8	Baxter icons used for the graphical interface (Freedman [2012]).	109
C.1	Action model used in experiments	162

Chapter 1

Introduction

Contents

1.1	Context	14
1.1.1	Cobotics	14
1.1.2	Robot Programming by Demonstration	15
1.1.3	Automated Planning	17
1.2	Problem Statement	17
1.3	Summary of contributions	19
1.4	Document organization	20

1.1 Context

Technologies have revolutionised the manufacturing industry ever since the Industrial revolution in the 19th century. Robots are improving productivity by replacing humans for arduous and repetitive manual tasks. Increasing order requests for industrial robots has led to higher capital investments into the field. Although robots can be superior in automating human tasks, there remain many tasks that cannot be completely taken over and still need human intervention, for example high-precision tasks. To allow both human precision and robot automation, collaborative robots have been introduced.

1.1.1 Cobotics

Collaborative robots, or *cobots*, have been introduced by Colgate and Peshkin [1999] to allow a close collaboration between humans and robots. They contribute to productivity gains as they are designed to take over manual and repetitive tasks. Cobots enable humans to perform tasks that they cannot perform on their own, due to physical constraints such as the manipulation of

heavy parts. Furthermore, they reduce risks of work-related accidents, including health hazards such as exposure to dangerous environments (*e.g.*, chemical acids, excessive temperatures or noise), as well as sleeping disorders caused by rotating work shifts. While replacing jobs of low-skilled human workers, they open up a market for new high-skilled jobs.

Cobotic systems have been adopted in several industries from the food-processing industry (Universal Robots [2016]), to aeronautics (Clapaud [2013]), to the health industry (Active8 Robots [2014]). However, companies resist the use of robots in their daily routines as they consider the investment cost ineffective, for example due to their high initial costs and the lack of trained personnel. Traditional robot programming solutions require domain experts who generally program robots to complete a specific task. Approaches using Neural Networks (Schmidhuber [2015]) or Reinforcement Learning (Gosavi [2009]) allow robots to learn on their own but become infeasible for some task-specific applications as they require large amounts of data. Many robot programming solutions fail as the deployment in real-world scenarios introduces further unexpected limitations. Thus, recent research has been focusing on robot programming for end-users where the robot learns from human teachers at the time of deployment.

1.1.2 Robot Programming by Demonstration

Robot 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 (Billard et al. [2008]). Influenced by natural learning paradigms in humans, it is an intuitive robot programming method with the goal to refine the robot's performance by providing repetitive demonstrations. PbD 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.

Figure 1.1 shows the life-cycle for teaching a robot by demonstration that consists of three main steps: 1. The teacher demonstrates a new skill to the robot (*e.g.*, by kinesthetically moving its arm), which the robot observes with its sensors. 2. The robot extracts the information such as relevant features to create a model of the skill and tries to apply it in a new context. 3. The execution is evaluated by the teacher who can refine the skill by providing additional demonstrations. The robot can generalise over multiple demonstrations by extracting common features. This incremental learning process allows non-robotics users to teach the robot new skills without having to write explicit code.

As the robot has limited knowledge about the world and restricted sensor availability, learning object manipulation tasks is still considered a hard problem (Ekvall and Kragic [2008]). Many PbD algorithms have been proposed in the literature (Argall et al. [2009], Billing and Hellström [2010]), but there still remain several challenges, such as the suboptimality of demon-

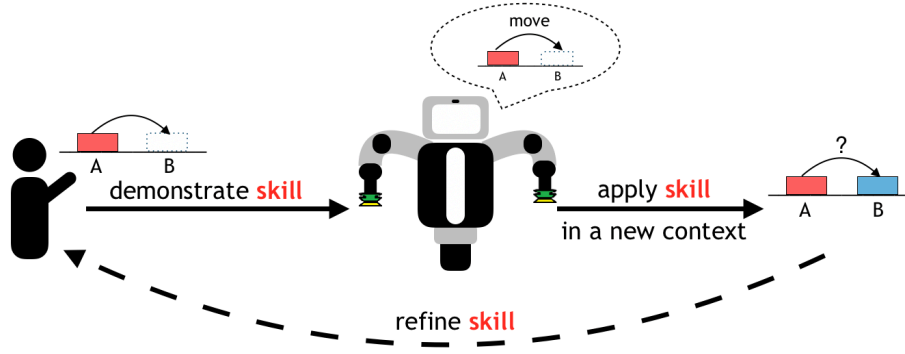


Figure 1.1: Overview of the Programming by Demonstration life-cycle consisting of three steps

strations (Chen and Zelinsky [2003], Kaiser et al. [1995]) or the lack of comparative user studies (Suay et al. [2012]). Another major problem is that the robot is generally demonstrated an action sequence to complete a specific task (Orendt et al. [2016], Peppoloni et al. [2014]), rather than individual reusable actions. Take for example the Tower of Hanoi problem (Hofstadter [1985]), a puzzle consisting of three pegs and a number of disks of different sizes stacked on one peg in descending order with the largest peg at the bottom (Fig. 1.2a) The goal is to move the entire stack from one peg to another, by obeying the following rules:

- move only one disk at a time,
- each move consists of taking the upper disk from one of the stacks and placing it on top of another (possibly empty) stack, and
- no disk can be moved on top of a smaller disk.

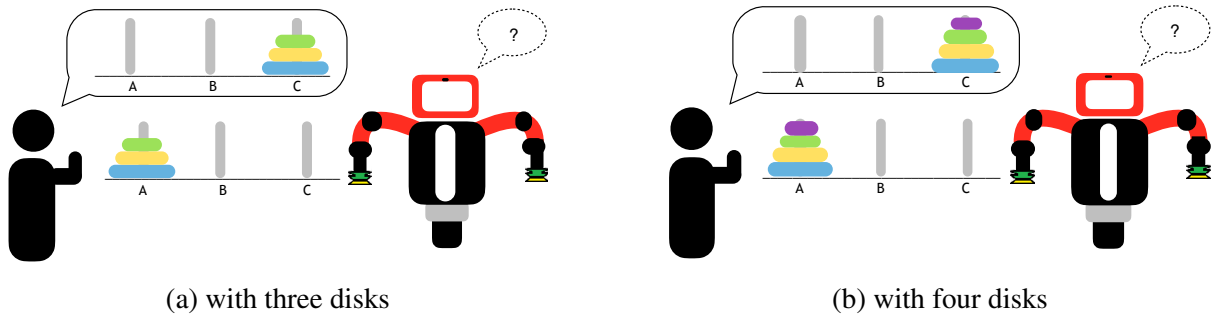


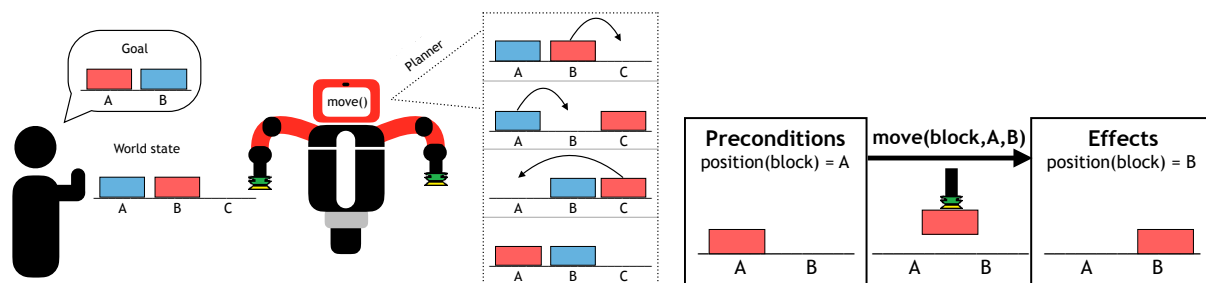
Figure 1.2: Tower of Hanoi problem where the goal is to move the entire stack of disks from one peg to another

The robot can be taught an action sequence to solve the problem for three disks. However, when the problem changes to four disks (Fig. 1.2b), the robot has to be demonstrated a new sequence as the solution is different depending on the given number of disks. In fact, the action sequence to solve this problem always consists of a combination of the same primitive action,

namely, to move a disk from one peg to another. 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.*, disks can only be placed on top of larger ones), and have the robot generate an optimal solution automatically, for example using task planners like in Automated Planning.

1.1.3 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 (Ghallab et al. [2004]). 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 and a goal, the task planner generates a sequence of actions, which guarantees the transition from the initial state to goal (Fig. 1.3a). To allow a correct transition between different world states, actions are defined in terms of preconditions and effects, which represent states before and after the action execution respectively (Fig. 1.3b). Planning algorithms use a symbolic planning language as their standard encoding language, such as STRIPS (Fikes and Nilsson [1971]) or PDDL (Ghallab et al. [2004]). 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, and 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.



(a) A planning problem can be solved if the robot is provided with a description of the world state, possible actions (*i.e.*, `move()`), and a goal. The planner can generate an action sequence to solve this problem.

(b) Actions are described in terms of preconditions and effects

Figure 1.3: Illustration of the main Automated Planning concepts

1.2 Problem Statement

In this thesis we argue for teaching robots primitive actions, instead of entire action sequences, and delegating the logical reasoning process of finding a solution to task planners. Most solu-

tions to teach the robot new actions assume a finite list of primitive actions that these actions are based on. However, to date, there is no database of general purpose primitive actions and whether a finite set can list all necessary motions is questionable (Billard et al. [2016]). While previous work has addressed integrating task planning with robotic systems (Abdo et al. [2013], Cashmore and Fox [2015]), enabling end-users to teach new actions from scratch that can be reused with task planners has not been addressed and is extremely challenging. The system should allow efficient programming of both *how* an action is performed (low-level action) and *when* it can be applied (high-level action), while being able to generalise both aspects to previously unseen scenarios. Furthermore, the system should provide an intuitive interface that allows end-users to reuse the taught actions together with a task planner, that is generally handled by domain experts. To that end, we must create a framework that combines the following aspects:

- **Programming by demonstration:** the user should be able to teach the robot primitive actions from scratch, including both *how* and *when* to perform the action, without writing explicit code,
- **Automated planning:** taught actions should be part of a planning domain that can be used by task planners to generate solutions,
- **Intuitive robot programming:** the user interface, planning domain and robot programming process need to be intuitive enough for users to understand easily and to allow them to navigate on their own in order to customise the robot to their specific tasks.

In other words, the system should enable the user to program the robot in an intuitive way, *e.g.*, by demonstration, and thereby construct reusable low- and high-level actions for the robot to accomplish previously unseen tasks. A functioning real-world system requires 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). The integration of all aspects into a real-time system is a major challenge. In this thesis we address this with the final goal to create a working end-to-end system for end-user robot programming. We address the following research question:

Can non-robotics users teach a robot actions from scratch that it can reuse to autonomously solve previously unseen problems?

With the focus on cobotic environments, we further investigate the difficulties encountered by end-users, when faced with PbD and Automated Planning concepts for the first time. We evaluate our approach with pick-and-place robots, addressing assembly and packaging tasks, two of the most common application types in manufacturing facilities (Robotic Industries Association [2018]).

1.3 Summary of contributions

This thesis contributes mainly to the fields of end-user robot programming and human-robot interaction, by proposing and implementing an end-user robot programming framework and conducting relevant user studies. The contributions can be summarised as follows:

- We propose iRoPro, an end-user robot programming framework that combines PbD and Automated Planning techniques, where the robot learns new actions from user demonstrations that it can reuse with a task planner. The robot programming process consists of the following steps:
 1. the user teaches the robot new actions by kinesthetic demonstration, where the robot learns both low- and high-level action representations,
 2. the robot uses these actions with a task planner to generate solutions to user-defined problems,
 3. the user can either have the robot execute the generated solution or revisit the taught actions via the graphical interface to refine them.
- User study 1: Experimental findings on issues encountered when non-robotics end-users are introduced to AI planning concepts and tasked to use a symbolic planning language to describe planning domains and problems to the robot. We evaluate the user's ability to construct action in terms of preconditions and effects and show that users with little to no programming experience can easily learn and use symbolic planning languages.
Research question: *How do non-expert users adopt the automated planning language with its action model representation?*
- User study 2: Experimental findings on user acceptance of the proposed framework. Using the Wizard-of-Oz technique, users were tasked to teach a robot actions by kinesthetic demonstration and assign conditions that can be used for automated planning.
Research question: *Can users teach a robot action models for automated planning using the robot programming framework?*
- Goal-oriented robot programming: We present a goal-oriented robot programming system for shelf organisation tasks using PbD and an graphical user interface. The system allows the robot to learn and execute organisation tasks from user input to simultaneously teach goals and actions by demonstration. The robot learns both *what* and *how* to perform a task using PbD and a goal inference model. We evaluated user teaching strategies with experiments on Amazon Mechanical Turk and compared their performance to eight benchmark strategies.

- We implement iRoPro as an end-to-end system on a Baxter research robot that
 1. allows simultaneous teaching of low- and high-level actions from a single demonstration,
 2. includes a user interface for action creation with condition inference and modification, and
 3. allows creating and solving previously unseen problems using a task planner for the robot to execute in real-time.

The implementation includes a graphical interface, which allows the user to teach new actions by kinesthetic demonstration, modify action conditions, define new planning problems for the robot to solve and execute the plan in real-time. This system enables end-users to program robots from scratch, without writing code.

- User study and system evaluation: We demonstrate iRoPro’s capability to generalise primitive actions on six benchmark tasks that are programmed and executed on the Baxter robot. We empirically investigate the usability of our system and validate its intuitiveness through a study with 21 users of different educational backgrounds and programming levels. 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.

1.4 Document organization

This thesis is organised into two parts:

The first part (I) provides a literature review of state of the art techniques in robot programming (Chapter 2), followed by an overview of Automated Planning (Chapter 3).

The second part (II) presents our contributions involving iRoPro, the proposed end-user robot programming framework (Chapter 4). We first conduct qualitative experiments and present experimental findings on user acceptance and issues encountered when introduced to Automated Planning concepts (Chapter 5). We then focus on evaluating our goal-oriented end-user robot programming approach and present our work on shelf organising tasks to simultaneous program goals and actions by demonstration (Chapter 6). Our main contribution is an end-to-end system implementation of iRoPro. We present details on the implementation and the system and user evaluation in Chapter 7.

Finally, we conclude this thesis in Chapter 8 and discuss possibilities for future work and research.

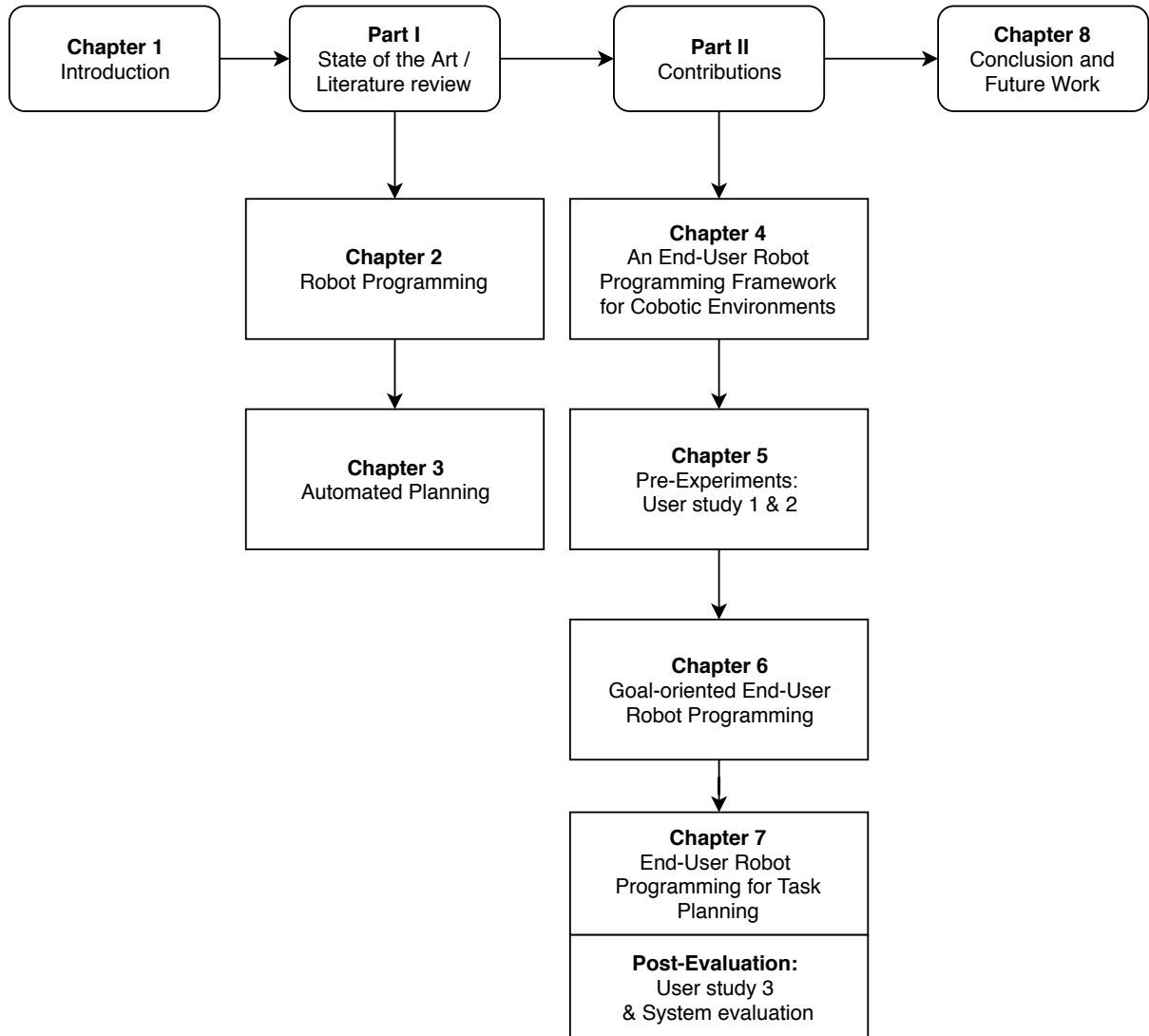


Figure 1.4: Overview of thesis chapters

Part I

Literature Review

Chapter 2

Robot Programming

Contents

2.1	Introduction	23
2.2	Manual Programming Systems	25
2.2.1	Text-based Systems	27
2.2.2	Graphical Systems	27
2.3	Automatic Programming Systems	27
2.3.1	Policy derivation techniques	29
2.3.2	Learning from data	31
2.3.3	Machine Learning Systems	31
2.3.4	Programming by Demonstration	32
2.4	End-user Involvement	34
2.5	Discussions and Relations to Present Work	35

In this chapter we first provide an overview of traditional robot programming techniques and categorise them to identify common bottlenecks. We discuss state-of-the-art techniques in manual (Sec. 2.2) and automatic (Sec. 2.3) programming systems and their required end-user involvement in the programming process (Sec. 2.4). We end this chapter by discussing common encountered issues and relations to this thesis (Sec. 2.5).

2.1 Introduction

Robot Programming is the process of defining desired motions and associated skills of the robot that it may perform without human intervention. Classical robot programming processes in the industry have task-specific definitions, which are generally robot-dependent, and require programming expertise. Figure 2.1 shows the classical robot programming process, consisting of

at least four phases: The initial definition of a resource-centered task and layout is followed by sequencing the workcell operations. Once the process has been validated, the robot is programmed offline using one or more native robot programming languages, before it is pushed into production for regular execution. The programmed robot can only be used for this specific task in this particular working environment. If the task definition needs to be modified, the programming expert has to repeat the entire programming process to ensure a consistently valid execution in production. Therefore, classical robot programming processes can be very time-consuming and cost intensive.

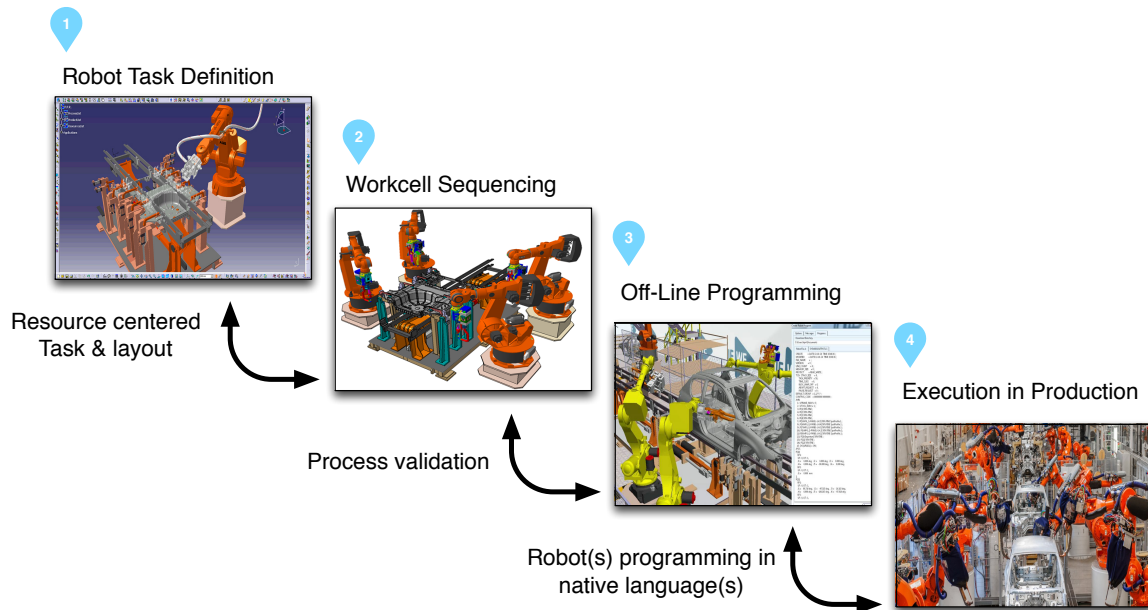


Figure 2.1: Classical robot programming process

In the past few decades, different robot programming techniques have been developed to facilitate this process. There are many ways to divide robot programming systems. Lozano-Perez [1983] divided them into three categories: guiding systems, where the robot's joint positions are sequentially recorded, robot-level systems, where a programming language is used, and task-level systems, where the task goal (*e.g.*, object positions) needs to be specified. However, the range of programming systems was very limited at that time and examined only industrial robot programming systems. Instead, Biggs and Macdonald [2003] divided them into two main categories, distinguishing programming methods between systems for programmers and users:

- **Manual programming:** the user can directly control the robot's execution code, using a text-based or a graphical system.
- **Automatic programming:** the user does not need to write explicit code and the robot learns using a learning or an instructive system.

Inspired by these main categories, we created an overview of the main robot programming techniques that have been applied in most recent research (Fig. 2.2). Even though many state-of-the-art systems use a combination of techniques, such as Deep Reinforcement Learning (Arulkumaran et al. [2017], Mnih et al. [2015]) or PbD for Reinforcement Learning (Hester et al. [2017]), it makes sense to differentiate them by their key attributes, which we identified as follows:

- **Control over the robot behaviour:** if the code created by the user directly encodes the robot's executed behaviour (manual vs automatic programming), and if created manually, how the code is generated (text-based vs visual programming)
- **Learning data:** for automated programming systems the robot learns from data, so we differentiate between the type of data used: if the data is biased or unbiased, if the teacher is involved in the learning process, how the data is acquired (provided by a teacher vs by self-exploration), and if it is labelled (supervised vs unsupervised learning)
- **End-user involvement:** the level that the end-user is involved in the programming process ranges from writing code manually, providing data or continuous feedback during the learning process to defining reward functions.

In the following sections we will give a brief overview of the identified programming system categories by first distinguishing between the user's control over the robot behaviour, *i.e.*, manual (Sec. 2.2) and automatic (Sec. 2.3) programming. For manual programming, we compare text-based (Sec. 2.2.1) and graphical (Sec. 2.2.2) systems. For automatic programming, we differentiate between those that learn from unbiased and biased data, *i.e.*, machine learning systems (Sec. 2.3.3) vs programming by demonstration (Sec. 2.3.4). Finally, we will discuss different levels of end-user involvement (Sec. 2.4) and conclude this chapter by discussing relations to this thesis (Sec. 2.5).

2.2 Manual Programming Systems

In manual programming systems, users have direct control over the robot code for which they often require expert knowledge in a programming language. There exists a variety of tools to make programming, as well as testing and debugging, easier such as IDEs, spreadsheets or macros. There are two types of manual programming: text-based programming, where the code is written manually in a chosen programming language (*e.g.*, python, C++, java) and graphical programming, where the code structure is created with the help of a graphical interface (*e.g.*, flow-charts).

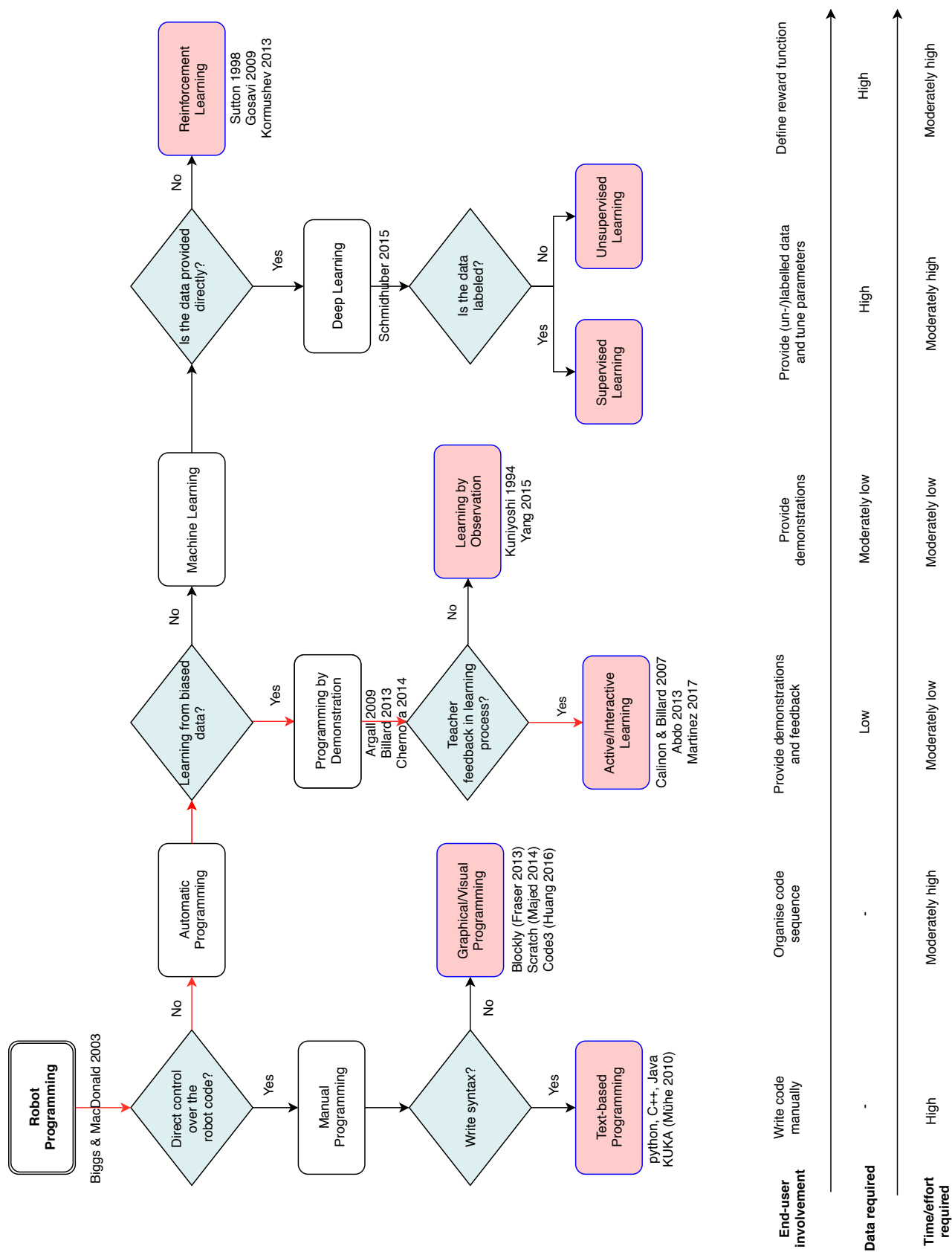


Figure 2.2: Overview of Robot Programming approaches based on control over robot behaviour code, type of learning data and how it is acquired and teacher feedback. Approaches are ranked by end-user involvement in the programming process as well as data and time required to learn a skill.

2.2.1 Text-based Systems

Text-based systems are one of the most common methods and use a traditional programming approach. Biggs and Macdonald [2003] differentiated between the type of programming language used: The user programs in either controller-specific languages, where the machine language consists of simple commands specific to the robot (*e.g.*, the KUKA Robot Language (Braumann and Brell-Cokcan [2011], Mühe et al. [2010]) shown in Fig. 2.3), generic procedural languages, where multi-purpose languages such as C++ have been extended with classes to provide simple access to common robot specific functions (*e.g.*, Lego [2003]), or behaviour-based languages that specify how the robot should react to different conditions (*e.g.*, Haskell (Hudak et al. [2002])). There has been a trend to move from low-level command-based languages towards more intelligent programming systems (*e.g.*, IDEs) with high-level languages that provide more support to the user and reduce the programming workload. However, text-based systems are more likely to be used by robot developers than end-users as they still require trained users with programming knowledge.

2.2.2 Graphical Systems

Graphical or icon-based systems use a graph, flow-chart or diagram view where users manually specify actions and program flow. The graphical icons correspond directly to predefined program statements. Lego [2003] and Bischoff et al. [2002] produced graphical systems using a flow-chart approach, where the robot's behaviour can be configured by arranging low-level actions in a sequence (Fig. 2.4). Similarly, Scratch (Majed [2014]) is a block-based visual programming language to allow children with no programming experience to learn development in an intuitive way for animations and interactive applications (Fig. 2.5). Blockly (Fraser [2013]) is a framework for building visual programming languages and has been used for programming mobile manipulator robots with a drag-and-drop interface as a high-level programming component (Huang et al. [2016]). While the user does not have to write code explicitly, they still have to manually sequence the components for the robot's behaviour. Thus, graphical systems still require some amount of programming effort and expertise to create the logical consistency of the robot program.

2.3 Automatic Programming Systems

Automatic programming systems relate to robots that can generate their behaviour from data provided as input to the system. Unlike manual programming, the end-user does not need to write or sequence robot code and does not have direct control over the robot's executed behaviour. The problem of learning a behaviour or *skill* can be considered as learning a function, referred to as a *policy*, that maps a world state to an action. In real-world applications, states

can only be partially observed due to restricted sensor availability. Hence, we assume that the robot learner has access to the observed state Z instead. A policy is a state-action mapping $\pi : Z \rightarrow A$ that allows the robot to select actions from an action domain A , given observations of the observed world state Z . States can either be represented in a discrete way (e.g., ‘object on the table’ or ‘robot holding object’) or in a continuous way (e.g., (x, y, z) coordinates of the object position)). Table 2.1 shows examples of the different representations for common state observations (e.g., object position, orientation, colour, spatial relation).

Table 2.1: State representations: continuous vs discrete

	Continuous representation	Discrete representation
Position	(x, y, z) coordinates	table
Orientation	$(\theta_x, \theta_y, \theta_z)$ angles	straight/upright
Colour	(r, g, b) values	red, yellow, green
Spatial relation	(x, y, z) distance vector	left/right/front/behind of

There are different techniques to derive the policy that represents the desired behaviour from the observed state. The most efficient policy derivation techniques approximate the state-action mapping from as few training data as possible. Chernova and Thomaz [2014] give an overview of state-of-the-art techniques for robots learning from human teachers and differentiate between policies that represent low-level *motions* and high-level *tasks* which use these motions.

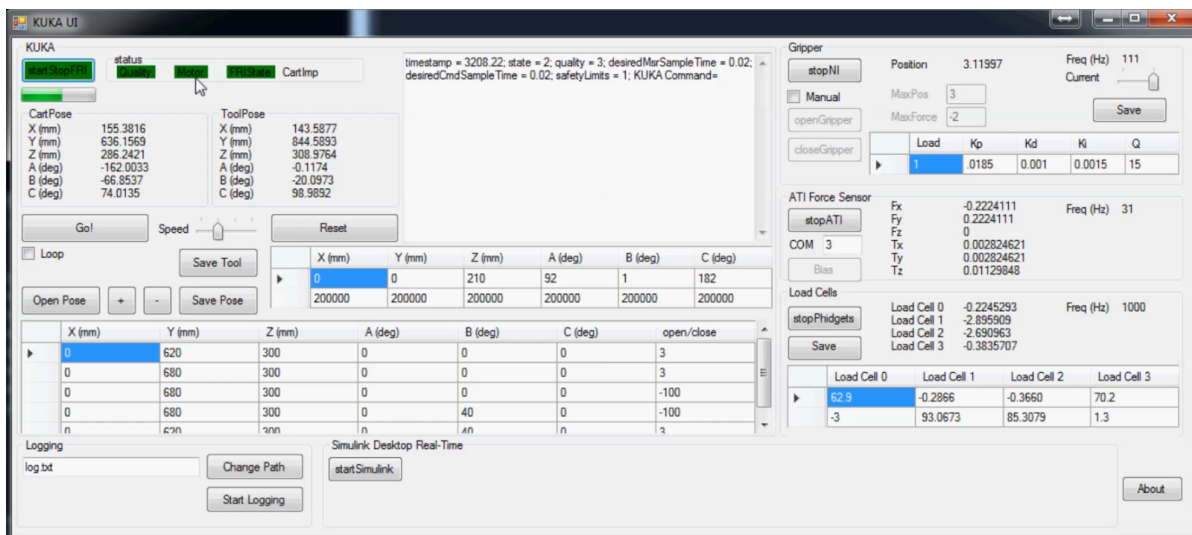


Figure 2.3: KUKA Robotics graphical user interface (Abdeetetal [2017])

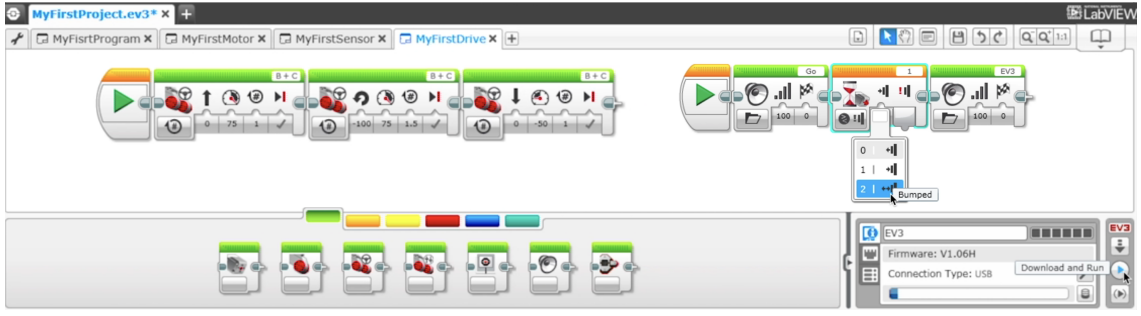


Figure 2.4: Lego Mindstorms EV3 icon-based interface (Lego [2003])

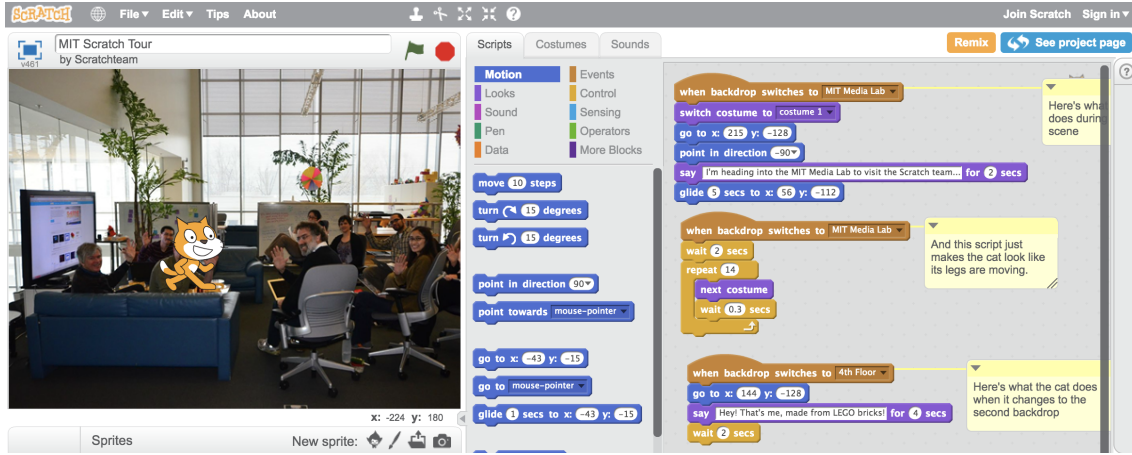


Figure 2.5: Scratch: block-based visual programming language (Majed [2014])

2.3.1 Policy derivation techniques

2.3.1.1 Low-level skill learning

Low-level representations focus on learning action trajectories or motions that can be generalised and applied to different tasks. In the literature low-level actions have been referred to as *skill*, *motor skill*, *primitive action*, or *low-level motion* (Chernova and Thomaz [2014]). Low-level actions can be learned from trajectory demonstrations using Gaussian Mixture Models (Billard et al. [2008]) or Dynamic Movement Primitives (Pastor et al. [2009]). They can also be learned from keyframe-based demonstrations, where the user kinesthetically manipulates the robot's arm to record a series of end-effector poses, referred to as *keyframes* (Akgun et al. [2012]). 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. Akgun et al. [2012] compares techniques to learn from trajectory with keyframe-based demonstrations and propose learning from hybrid demonstrations that is suitable for learning any type of skill (Fig. 2.6). Alexandrova et al. [2014] 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 user interface (Fig. 2.7).

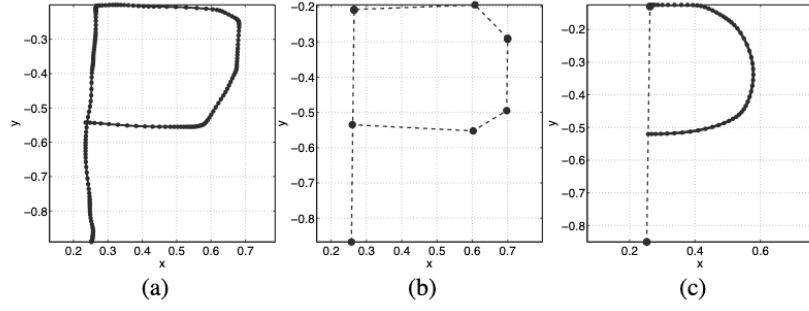


Figure 2.6: Sample demonstrations of the letter P in 2D: (a) trajectory demonstration (b) keyframe-based demonstration (c) hybrid demonstration (Akgun et al. [2012]).

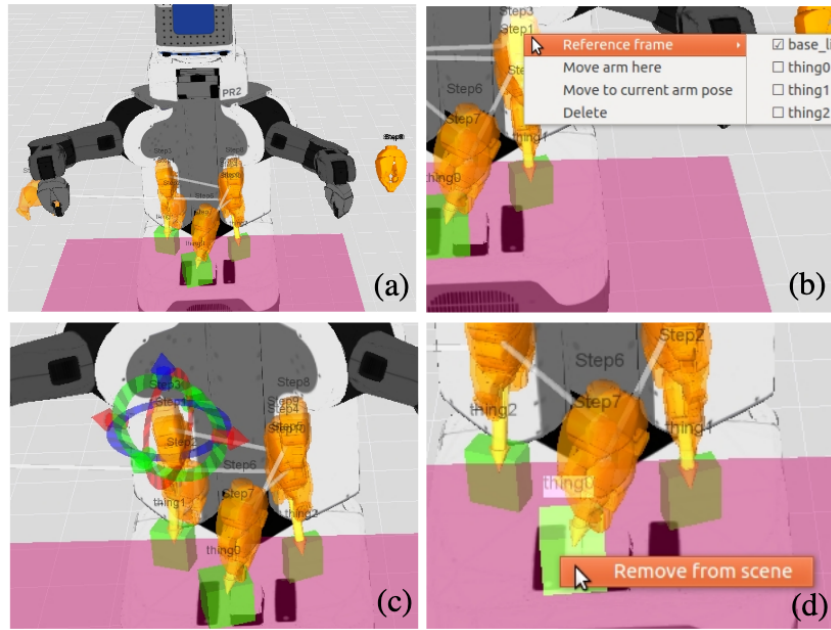


Figure 2.7: Robot programming by keyframe-based demonstration using a graphical user interface that allows retrospectively editing (Alexandrova et al. [2014])

2.3.1.2 High-level task learning

High-level task learning focuses on learning the *sequence* of primitive actions, *i.e.*, the order to execute them. Primitive actions are generally preprogrammed into the robot (Peppoloni et al. [2014]), where high-level tasks are learned from observing complete task executions and extracting the primitive actions. Paxton et al. [2017] represent task plans with Behaviour Trees that users can modify manually to adapt to other tasks. She et al. [2014] teaches the robot new high-level tasks by instructing the robot a sequence of lower-level actions. Rather than representing the new action as an action sequence, it is modelled by their desired goal states obtained from object states at the beginning and at the end of the instruction. Ahmadzadeh et al. [2015] learns the task goal by inferring preconditions and effects from multiple demonstrations.

2.3.2 Learning from data

There are different ways to learn from data depending on the amount provided at a time (incremental vs batch learning) and the type of data. The robot can learn from data that is directly related to the desired behaviour and has been selected by a teacher (*e.g.*, from optimal teacher demonstrations), which we refer to as learning from *biased* data. If the robot learns from data that is not directly related to its executed behaviour (*e.g.*, from trial and error), we refer to it as learning from *unbiased* data. Thus, we differentiate automatic programming systems between Machine Learning (Sec. 2.3.3), where the robot learns from *unbiased* data, and Programming by Demonstration (Sec. 2.3.4), where the robot learns from *biased* data. In Machine Learning, the data can be provided directly to the robot (*e.g.*, Deep Learning) or acquired by self-exploration of the environment (*e.g.*, Reinforcement Learning). Recent approaches have combined multiple techniques to improve the performance and accelerate the learning process, *e.g.*, Deep Reinforcement Learning (Arulkumaran et al. [2017]) or PbD for Reinforcement Learning (Hester et al. [2017]). As this is beyond the scope of this thesis, we will give a brief overview of the main techniques and leave their discussion for future work.

2.3.3 Machine Learning Systems

Machine Learning (ML) systems use inductive inference to learn a policy from unbiased data, which includes data unrelated to the desired robot behaviour. The goal of these systems is to construct programs that allow the robot to automatically improve its performance with increasing amounts of data. Even though ML algorithms have been around since the 1980s, it has only become popular in the past few decades. The rise of the internet led to Big Data, improved knowledge sharing, and advances in techniques to process and store data efficiently triggered a wave of new ML techniques. Recent applications of ML in robotics (Faggella [2019]) include research areas such as Computer Vision (or Robot Vision) for the identification and sorting of objects (Stager et al. [2013]) and Imitation Learning to learn action plans from watching unconstrained videos (Yang et al. [2015]). Many approaches derive a policy using vision by having the robot learn from image or video data or by observing a human teacher (Kuniyoshi et al. [1994]).

We differentiate between the provided input data (Fig. 2.8): In Deep Learning (DL) (Schmidhuber [2015]), data is provided directly to the robot in either labelled (Supervised learning) or unlabelled format (Unsupervised learning). DL systems generally use artificial Neural Networks (NN) which require large amounts of data and computation power to learn the desired behaviour. Techniques are divided into supervised (*e.g.*, Classification, Regression) and unsupervised (*e.g.*, Clustering) learning, using labelled and unlabelled data respectively.

In Reinforcement Learning (RL) (Gosavi [2009], Kaelbling et al. [1996], Sutton et al. [1998]) the robot acquires data autonomously by exploring its environment. The robot enters

different states depending on the actions it takes and learns from the observed state changes. While taking random actions, the robot uses a reward function to learn its policy and refines it by updating expected rewards with observed outcomes. However, setting up the system's states, actions and initial policy is often difficult and usually requires robotics experts. Kober et al. [2013] provides a survey of RL techniques to generate robot behaviours and highlights the main challenges that are faced. Since the robot has to explore many states before it can learn the correct behaviour, RL systems usually take hundred or thousands of training examples which are usually collected in a simulated environment. To accelerate the learning and reduce the amount of exploration required, recent work includes teacher demonstrations with RL solutions (Hester et al. [2017], Martínez et al. [2017]).

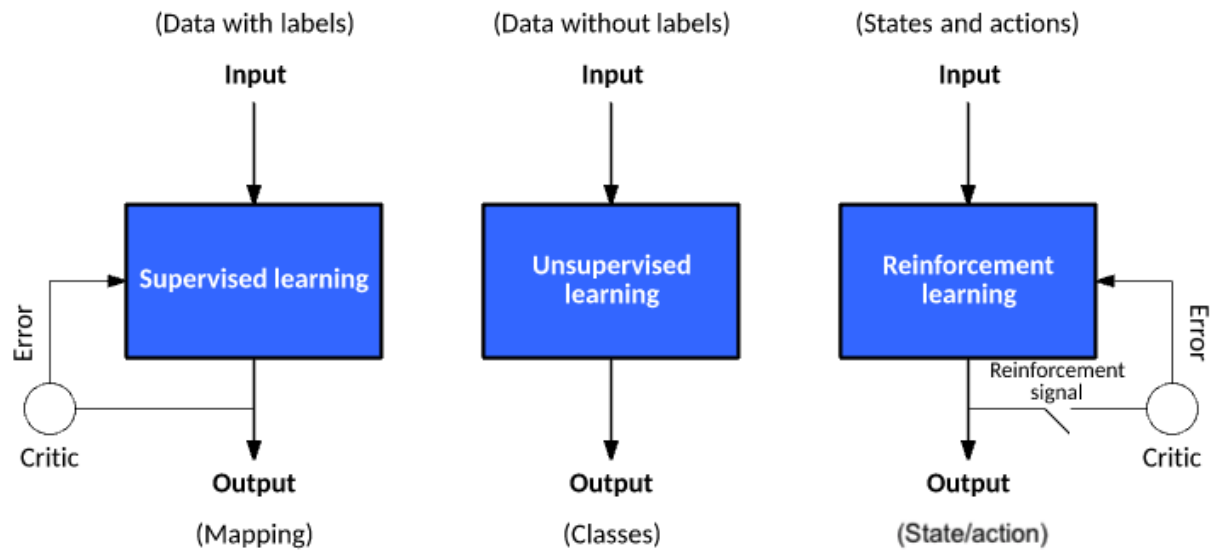


Figure 2.8: Types of Machine learning: Deep learning (supervised and unsupervised learning) and Reinforcement learning (Jones [2017])

2.3.4 Programming by Demonstration

Programming by Demonstration (PbD) (Argall et al. [2009], Billard et al. [2016]) describes various techniques where the robot learns new behaviours from teacher demonstrations. In the literature, it has also been referred to as *Learning from Demonstration*, *Imitation Learning*, *Learning by Showing*, *Learning from Observation*, *Behavioral Cloning* or *Mimicry*. It provides an intuitive medium to allow non-roboticists to communicate skills to robots without having to write code. Unlike ML solutions, the provided data is biased, *i.e.*, directly correlated with the robots executed behaviour (*e.g.*, human teachers providing optimal demonstrations). The underlying concept is for the robot to learn a new skill more efficiently from these selected demonstrations, thus reducing the complexity of the search space and accelerating learning. As the teacher selectively provides demonstrations that the robot should learn from, the data is

generally sparse and research has focused on learning from as few demonstrations as possible. The user can choose to provide positive or negative examples (Grollman and Billard [2012]). Positive examples are demonstration data showing what the robot should do and what it can learn directly from. Negative examples are what the robot should *avoid* and help to generalise faster by eliminating bad solutions from the search space. Walsh [2010] noted that negative examples are highly uninformative as the learner cannot easily determine the reason for the failure, while positive examples provide very useful information as a superset of the literals that belong to the action preconditions is identified.

There are different techniques to learn a policy depending on how demonstrations are provided to the robot and the chosen interaction modalities. The robot can learn from simply observing the teacher demonstration (Learning by Observation). As human demonstrations are often noisy or suboptimal, interactive policy refinement techniques can be used, where robots learn from continuous feedback provided by the teacher (Active or Interactive Learning).

2.3.4.1 Providing demonstrations

There exist various techniques for providing demonstrations. Argall et al. [2009] define four categories by differentiating between the choice of the demonstrator (human or robot) and who executes the demonstration: The teacher could demonstrate either using their own hands (*shadowing* or *external observation*), by wearing sensors (*sensors on teacher*), or by moving the robot's joints directly (*kinesthetic teaching* or *teleoperation*). If the demonstration is not recorded directly on the robot's joints (*e.g.*, shadowing), the demonstrated trajectory has to be extracted and mapped to the robot's joints. This is related to the correspondence problem (Nehaniv et al. [2002]), which describes the difference of humans and robots regarding their sensing abilities and physical embodiment. If the robot joints are recorded directly from the demonstration (*e.g.*, kinesthetic teaching), this problem is eliminated. Previous work showed that kinesthetic guidance can be problematic in narrow spaces (Wrede et al. [2013]) or if the objects are large or dangerous to operate (Chernova and Thomaz [2014]). However, comparing kinesthetic teaching with teleoperation showed that kinesthetic teaching produced better results in terms of efficiency, effectiveness (success and error rate), and usability (Akgun and Subramanian [2011], Chernova and Thomaz [2014], Fischer et al. [2016]).

2.3.4.2 Interaction modalities

There are different ways for the teacher to transfer the data to the robot, for example using voice, vision, touch or gestures. PbD systems often use touch to provide demonstrations, *i.e.*, by kinesthetically moving the robot's joints. Voice recognition can be used during demonstrations to activate in-built actions such as opening gripper or saving end-effector poses (Alexandrova et al. [2014]). Such instructive systems provide the user with a high-level control to program the

action sequence by commanding sub-actions that have already been programmed into the robot (Forbes et al. [2015]). Multi-modal communication that combines information from vision, gesture and voice sources can be used to clarify instructions to the robot, for example mentioning ‘that table’ and gesturing the relevant object at the same time. Profanter et al. [2015] evaluated four input modalities (touch, gesture, speech, 3D tracking device) and showed that users preferred touch and gesture input, while speech input was the least preferred modality.

2.3.4.3 Interactive policy refinement techniques

For active, or *interactive*, learning, the teacher is involved in the learning process by providing regular feedback to the learned action (Calinon and Billard [2007a,b], Nicolescu and Mataric [2003]). The robot first observes the demonstration performed by the teacher. When it reproduces the action, the teacher can improve and correct the movement by physically moving its limbs. In Nicolescu and Mataric [2003] the robot refines the learned skill using feedback cues provided by the teacher and by inserting or removing behaviours from the network of abstract behaviours. Learning tasks from interactions with a teacher is also known as *Interactive Task Learning* (Laird et al. [2017]), where the goal is to learn through natural communication and to improve performance via instruction, demonstration, and feedback. The robot can also request feedback from the teacher when encountering problems during action execution (Abdo et al. [2013], Cakmak and Lopes [2012], Martínez et al. [2017]). Other policy refinement approaches allow the teacher to modify the taught actions using a graphical interface, therefore minimising the number of demonstrations required. (Alexandrova et al. [2015], Paxton et al. [2017], Perzylo et al. [2016], Stenmark et al. [2017]).

2.4 End-user Involvement

There are different levels of end-user involvement in robot programming, such as writing and organising the execution code in manual programming or providing data in the form of demonstrations, labelled or unlabelled data for automatic programming systems. Inspired by Kormushev et al. [2013] who compared different robot teaching approaches, Table 2.2 compares the presented robot programming approaches by user involvement task, expertise required, as well as data and time required to learn a skill. For the last three categories, we assign a ranking from ‘high’, ‘moderately high’, to ‘moderately low’ and ‘low’. The required expertise corresponds to how hard it is for an end-user without programming experience to complete the user tasks for the programming approach.

While manual programming approaches do not require any data, they can be considered most difficult and time consuming, as the user has to write or sequence the execution code manually. Even though visual programming facilitates the programming experience over text-

based programming, it still requires users to have a good understanding of the constructed program flow.

Programming by demonstration (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. While it does not require programming experts, the difficulty lies in providing correct examples that are diverse enough to allow the robot to learn a generalised skill that is applicable to new scenarios.

Other automated programming approaches, such as Reinforcement Learning (RL) and Deep Learning (DL), allow the robot to learn autonomously, but require programming and domain experts to prepare the system input (*e.g.*, label or preprocess data for DL, define policy and reward functions for RL). These systems generally require vast amounts of data and computation power to learn a skill and are usually trained in simulation.

Table 2.2: End-user involvement for common robot programming approaches

Programming approach	Tasks involved	Expertise required	Data required	Time required to learn skill
Text-based	manually write code	high	-	high
Visual/graphical	manually sequence code	moderately high	-	high
Programming by demonstration	provide demonstrations	moderately low	moderately low	moderately low
Reinforcement learning	specify reward function, policy parameterization, initial policy, <i>etc.</i>	moderately high	high	moderately high
Deep learning	preprocess data, tune parameters	moderately high	high	moderately high

2.5 Discussions and Relations to Present Work

Robots are currently being used for many industrial applications from welding (*e.g.*, arc, spot welding) to material handling (*e.g.*, pick-and-place, packaging, palletizing) (Technavio [2018]). Furthermore, there exists a wide range of robot grippers (*e.g.*, claw, suction, magnetic grippers) that are dependent on the robot platform. 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. A feasible programming system

could involve a human operator teach the robot new actions and have the robot automatically generalise the taught action to new scenarios.

Manual programming systems provide direct control over the robot’s behaviour, but require users with programming expertise and involve high programming time and effort for debugging and testing. The solutions are often task-specific and cannot be changed easily by end-users. Automatic programming systems require significantly less effort when it comes to the programming process. However, DL solutions generally need large amounts of training data that is relevant for the desired application. RL solutions can be time-consuming as they require the robot to explore the environment and collect the necessary data to learn an optimal policy. There is on-going research to reduce the amount of data and time required to learn an optimal policy with DL or RL systems, but the involved tasks (*e.g.*, defining the reward function or tuning parameters) remain difficult for end-users without programming experience.

Programming by Demonstration (PbD) allows end-users to teach the robot new tasks by taking demonstrations as input and inferring the policy for the task. An intuitive way to provide demonstrations is to kinesthetically manipulate the robot’s arms, followed by refining the learned action with subsequent demonstrations or modifications on a graphical interface. This allows users to teach the robot new actions with minimal programming effort and independent of the robot’s architecture. However, existing PbD solutions usually require users to teach robots an entire action sequence to achieve a certain goal. If the goal changes, the user has to find a new solution and teach the robot again. In this thesis we argue for teaching robots primitive actions, instead of entire action sequences, that can be used with task planners. The idea is to associate semantic meanings to actions in order to delegate the logical reasoning process of finding a solution to task planners. For example, when teaching a pick-and-place action, the robot should be taught the semantic meaning in the form of high-level conditions for executing the action (*e.g.*, to grab an object only if the gripper is empty). This information allows the robot to reuse and apply the taught actions in a different context. We have identified two main aspects that we address in this thesis:

1. **Teaching reusable actions from scratch.** Existing PbD implementations are task-specific and cannot be applied to previously unseen scenarios. Many PbD implementations already have pick and place actions coded into the system (Veeraraghavan and Veloso [2008]) and the robot is taught an action sequence to achieve a predefined goal. Even slight changes in the goal could require a different solution, so the robot needs to be taught a new action sequence again. This highlights a key issue in PbD, namely, to design a system that allows the robot to learn generic tasks that are applicable to different scenarios. In this work, we assume that the robot does not have any primitive actions preprogrammed. We want to enable the user to teach reusable actions from scratch.
2. **Automatic action sequence generation.** After having learned all primitive actions, the

robot should be able to reuse them to complete previously unseen tasks. Current solutions for high-level task learning either teach the robot entire action sequences or include an intermediate manual step where the user has to manually sequence actions. Finding an optimal solution can be tedious, expensive, or even computationally impossible for humans (*e.g.*, solving a rubik's cube). Instead, we want to delegate this logical reasoning process of finding a solution to task planners, thus facilitating the programming process for the human operator.

In the following Chapter 3 we will give an overview of Automated Planning.

Chapter 3

Automated Planning

Contents

3.1	Introduction	39
3.2	Classical planning	40
3.3	Planning Domain Definition Language (PDDL)	42
3.3.1	Planning domain description	43
3.3.2	Planning problem description	45
3.3.3	PDDL evolution and extensions	46
3.4	Planners	47
3.5	Knowledge Engineering Tools	49
3.6	Discussions and Relations to Present Work	49

In this chapter we give a brief overview on Automated Planning and its main concepts. We first present the theoretical definitions (Sec. 3.2) and the standard encoding language used (Sec. 3.3). We then discuss common techniques for planning (Sec. 3.4) and knowledge engineering tools for modelling planning domains (Sec. 3.5). We conclude this chapter by discussing relations to this thesis (Sec. 3.6).

3.1 Introduction

Automated planning, also known as AI planning or task planning, is an area in A.I. that studies the deliberation process of choosing and organising actions to achieve a goal (Ghallab et al. [2004]). Humans automatically anticipate the outcome of their actions, even if they are not fully aware of it. Automated planning techniques try to efficiently reproduce human reasoning and behaviour and can be used to model a robot's skills and strategies, when operating in diverse environments, without the need for expensive hand-coding.

The focus in automated planning lies within the development of domain-independent planning systems, called planners or *task planners*. These planners consist of search algorithms which are not problem-specific and can generate solutions to previously unseen problems. Given a description of the state of the world, a set of actions, an initial and a goal, the planner generates an ordered sequence of actions, which guarantees the transition from the initial state to the goal. Actions are defined with *preconditions*, *i.e.*, conditions on the state of the world in order to execute the action, and *effects*, *i.e.*, changes in the state of the world after the action execution. Planning can be considered the process of choosing appropriate actions to bring the state of the world to a target state. In the following section we will present the main definitions used in classical planning.

3.2 Classical planning

In classical planning, world dynamics are modelled as state transition systems. We will start by providing some definitions of general concepts used in this thesis that have been derived from Ghallab et al. [2004].

Definition 3.2.1. A *state transition system* is a triple $\Sigma = (S, A, \gamma)$ such that:

- $S = \{s_1, s_2, \dots\}$ is a finite set of states,
- $A = \{a_1, a_2, \dots\}$ is a set of actions,
- $\gamma : S \times A \rightarrow S$ is a state transition function.

A state transition system Σ is represented as a directed graph where nodes are states of S and arcs are actions of A (Fig. 3.1). Applying an action a to state s produces a new state $s' = \gamma(s, a)$ that is represented as an arc from s to s' , referred to as a *state transition*. Σ is deterministic if for all states s and actions a and the transition function $\gamma(a, s)$ produces a unique state s' .

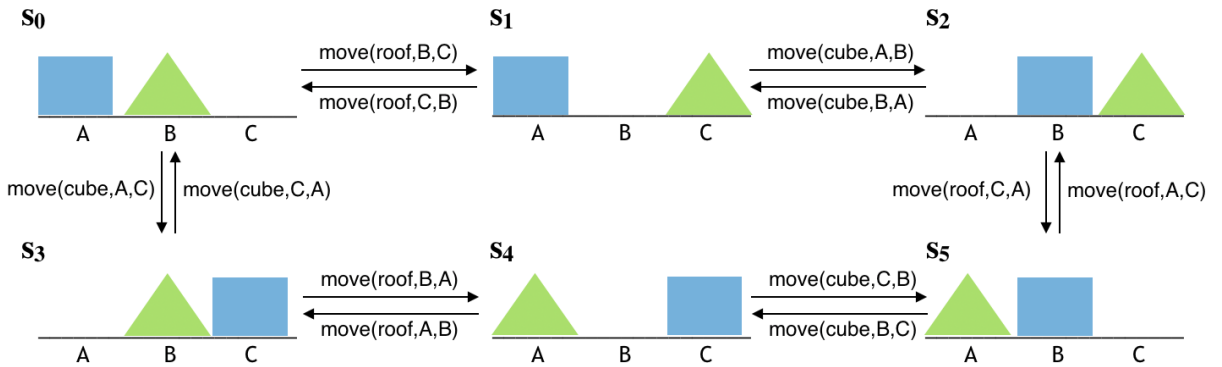


Figure 3.1: A state transition system consisting of six states s_0 to s_5

Logical representations are one of the most commonly used representations for classical planning problems. Each world state s is represented by a set of logical propositions p , denoting facts of the world that are *true*. If p is not in the state s , it is considered to be *false*. *Planning operators* and *actions* change the world state by modifying the truth values of the propositions and are represented in terms of *preconditions* and *effects*.

Definition 3.2.2. A *planning operator* o is a tuple $o = (\text{name}(o), \text{precond}(o), \text{effect}(o))$, whose elements are as follows:

- $\text{name}(o)$ is the *name* of the operator,
- $\text{precond}(o)$ is a set of literals that must be true to apply the operator o ,
- $\text{effect}(o)^-$ is a set of literals that are false after the application of the operator o ,
- $\text{effect}(o)^+$ is a set of literals that are true after the application of the operator o ,

where $\text{effect}(o) = \text{effect}(o)^- \cup \text{effect}(o)^+$ and $\text{effect}^-(o) \cap \text{effect}^+(o) = \emptyset$. An operator has a set of parameters that are associated with a *type* which can be part of a type hierarchy with multiple layers of subtypes. For example, a general type *ELEMENT* with subtypes *OBJECT* and *POSITION*, and *BASE*, *CUBE*, *ROOF* objects (Fig. 3.2).

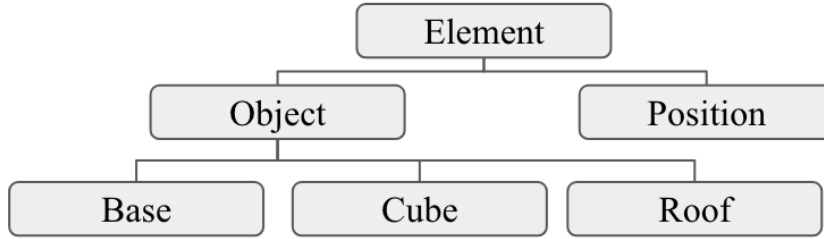


Figure 3.2: Example of a type hierarchy showing a general type *ELEMENT* with subtypes *OBJECT* and *POSITION*, and *BASE*, *CUBE*, *ROOF* objects.

Definition 3.2.3. An *action* is any ground instance of an operator. Or reversely, we consider an operator as a *generalised action*. If a is an action and s is a state such that $\text{precond}(a)$ are true in s , then a is *applicable* to s and applying action a to state s results in a state s' , such that:

$$s' = \gamma(s, a) = (s - \text{effects}^-(a)) \cup \text{effects}^+(a), \quad (3.1)$$

Definition 3.2.4. A *planning domain* is a state transition system $\Sigma = (S, A, \gamma)$, with the state transition function γ as stated in Eq. 3.1.

Definition 3.2.5. A *planning problem* is a triplet (Σ, s_0, g) where:

- Σ is a planning domain,

- $s_0 \subseteq S$ is the initial state of the world,
- g is a set of goal propositions describing the goal to achieve.

Definition 3.2.6. A *plan* is any sequence of actions $\pi = \langle a_1, \dots, a_k \rangle$, where $k \geq 0$.

The state produced by applying π to a state s is the state obtained by applying each action of π sequentially. We can denote this by extending the state transition function to plans as follows:

$$\gamma(s, \pi) = \begin{cases} s, & \text{if } |\pi| = 0 \\ \gamma(\gamma(s, a_1), \langle a_2, \dots, a_k \rangle), & \text{if } |\pi| > 0 \text{ and } a_1 \text{ is applicable to } s \\ \text{undefined,} & \text{otherwise.} \end{cases}$$

A state s_n is reachable from a state s_0 , if there exists a plan π such that $s_n = \gamma(s_0, \pi)$. When a full specification (*i.e.*, planning domain and problem) are provided to the planner, it generates a plan to achieve the specified goal using the available actions. Thus, a plan π is a *solution* to a planning problem if it guarantees the transition from the initial states s_0 to the goal states, *i.e.*, $g \subseteq \gamma(s_0, \pi)$.

When actions are triggered, they change the world state according to their effects and are not necessarily reversible. Actions are not combinable in any order and have precedence constraints. Consider again the Tower of Hanoi problem, where the disks need to be stacked in ascending order (smallest at the top). The two last actions consist of stacking the second smallest disk and then the smallest disk on the peg. Reversing the order will result in a violation of the rule that larger disks cannot be stacked onto smaller ones. Thus, plans need to be performed in the correct order to achieve the goal.

Furthermore, the progression towards the goal is not always monotone, as actions can also have negative effects. In the example with the Tower of Hanoi problem, our goal state is to have “ $disk_n$ is on top of $disk_{n+1}$ ” and “ $disk_{n+1}$ is on the table” for a problem with $n + 1$ disks. If in the initial state the disks are stacked in ascending order on the left peg, then moving the top-most $disk_1$ to any other peg will delete the fact “ $disk_1$ is on top of $disk_2$ ”, and will have to be added again later to fulfil the goal.

3.3 Planning Domain Definition Language (PDDL)

Originally developed by Ghallab et al. [1998] and the Planning Competition Committee, the Planning Domain Definition Language (PDDL) has become the standard encoding language for classical planning. PDDL expresses the ‘physics’ of a domain, *i.e.*, available predicates, possible actions and their effects. It supports several syntactic features including conditional effects, specification of safety constraints and hierarchical actions composed of subactions and

subgoals. For example, in manufacturing environments, PDDL can be used to formalise the configuration of available resources together with the intended goal in order to find a solution using task planners (Huckaby et al. [2013]).

3.3.1 Planning domain description

A PDDL planning domain consists of objects and their types, predicates describing the object states and possible actions. As a running example we will consider a planning domain *iRoPro* to describe pick-and-place tasks with three types of objects (*BASE*, *CUBE*, *ROOF*). The planning domain description is given as follows:

```
(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))
  )
)
```

A domain description always starts with the declaration of its name, *e.g.*, *iRoPro*, preceded by the keyword `:domain`. The requirements (identified by `:requirements`) allow to characterise the expressiveness and abstraction levels of the domain, *e.g.*, `:strips` indicates the use of STRIPS (Fikes and Nilsson [1971]), the most basic subset of PDDL, `:typing` indicates that the domain can declare parameter and object types. Other requirements include `:equality` to use the predicate ‘=’ as an equality term or `:adl` to use conditional and universal quantifier terms.

`:types` describes the type names and hierarchy that can be used subsequently for predicates or actions. In this domain, we defined a 3-layered type hierarchy consisting of a general type *ELEMENT*, subtypes *POSITION* and *OBJECT*, with *BASE*, *CUBE*, and *ROOF* (Fig. 3.2). `:predicates` are logical symbols to specify facts about objects that can be used for actions. Predicates can be standalone or have parameters with specified types. Each parameter name is preceded with a question mark ‘?’, followed by a ‘-’ and its type name. For instance, `(on ?o - object ?e - element)` expresses that a variable `?o`, whose type is `object`, is placed on top of an element `?e`. The example planning domain is comprised of the following predicates, described as follows:

Table 3.1: Predicates for the iRoPro domain

Predicate in PDDL	Description in English
<code>(clear ?e - element)</code>	an element has nothing on top of it
<code>(thin ?o - object)</code>	an object is thin
<code>(flat ?e - element)</code>	an element has a flat top
<code>(on ?o - object ?e - element)</code>	an object is on an element
<code>(stackable ?o - object ?e - element)</code>	an object can be placed on an element

Note that `thin` can only be applied to an object type, while `flat` can refer to any element type. Furthermore, `on` describes the spatial relation between two parameters and `stackable` describes the *possibility* of placing them on top of each other.

Finally, the keyword `:action` defines a planning operator as in Definition 3.2.2. In general, an action declaration consists of the following:

- `:parameters` – variables which it applies to,
- `:precondition` – conditions that must be satisfied before it can be executed; if none are specified, it is always executable,
- `:effect` – changes in the world state imposed after the execution.

In the example domain, the `move` action describes moving an object (`?obj`) from position A to B, if both the object itself and the target position (`?B`) are clear. After the `move` action execution, the effects express that the object is no longer at initial position `?A` but on `?B` and that `?A` is clear, but `?B` is not (Fig. 3.3).

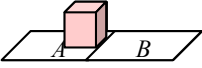
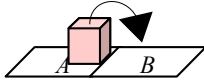
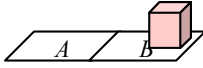
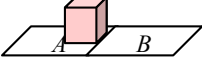
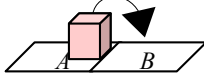
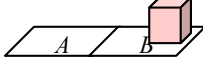
Preconditions: (at cube A) (empty B) 	(move cube A B) 	Effects: (at cube B) (empty A) 
Preconditions: (at ?obj ?posA) (empty ?posB) 	(move ?obj ?posA ?posB) 	Effects: (at ?obj ?posB) (empty ?posA) 

Figure 3.3: Action model representation of a move action in terms of preconditions and effects: an action (or instantiated operator) for a cube (top), and generalised action (or planning operator) for any object, where variables are prefixed with ‘?’ (bottom).

3.3.2 Planning problem description

A planning problem consists of an initial state and a goal to be solved using actions in the associated planning domain. For example, the problem of swapping two cube objects *obj1* and *obj2* on positions A and B respectively, with a third position C unoccupied, is described as follows:

```
(define (problem swap)
  (:domain iRoPro)
  (:objects
    obj1 obj2 - cube
    A B C - position)
  (:init (on obj1 A) (on obj2 B) (clear C))
  (:goal (on obj1 B) (on obj2 A) )
)
```

A planning problem description always starts with the declaration of its name, *e.g.*, *swap*, preceded by the keyword `:problem`. `:domain` specifies the domain it operates in, *e.g.*, *iRoPro*. `:objects` lists all instantiated objects in the world state with their names (Fig. 3.4b). This world is composed of three positions, A, B and C, and two objects, *obj1* and *obj2*, which are both of type *cube*. `:init` describes the initial world state of *obj1* and *obj2* on positions A and B respectively, and C unoccupied. The goal is to find a plan to swap the two objects, *i.e.*, *obj1* from A to B and *obj2* from B to A (Fig. 3.4d). The planner can generate one of the two action sequences shown in Table 3.2. Even though both plans consist of different actions, they both lead to the same goal states (similar to Fig. 3.1). Another planning problem

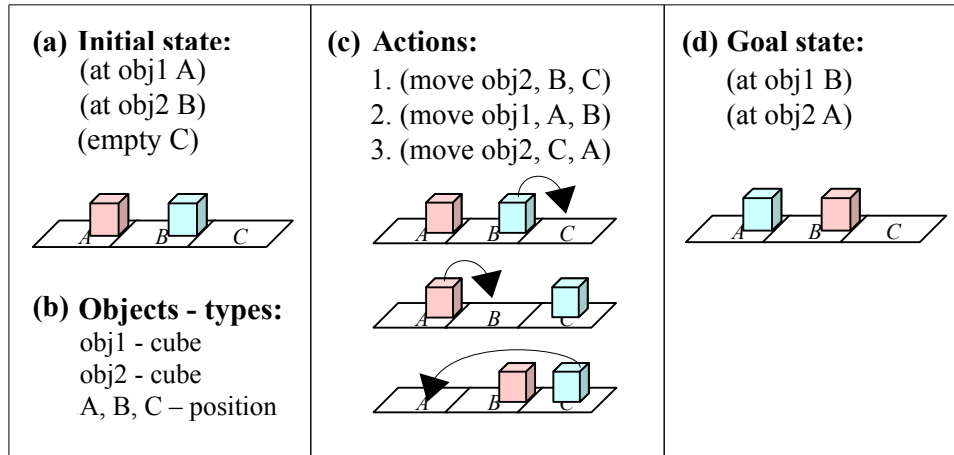


Figure 3.4: Definition of a planning problem: (a) properties describing the initial world state (b) object names and their types (c) instantiated actions (d) properties describing the goal state

Table 3.2: Two plans generated by the planner to swap the positions of two objects (Fig. 3.1).

Plan 1 (Fig. 3.4c):	Plan 2:
1. move(obj2, B, C)	1. move(obj1, A, C)
2. move(obj1, A, B)	2. move(obj2, B, A)
3. move(obj2, C, A)	3. move(obj1, C, B)

in this domain could include multiple objects of different types which need to be stacked in a given order, such as the Tower of Hanoi problem. As we can define an infinite number of objects in our planning domain, there exists an infinite number of planning problems that can be defined and solved. PDDL provides a means to model and solve problems in any real-world environment in the form of a planning domain.

3.3.3 PDDL evolution and extensions

Since the first version PDDL 1.2 (Ghallab et al. [1998]) was introduced as the official language of the 1st International Planning Competition (IPC), there have been several new versions and extensions. Succeeding versions that allow the representation of real-world problems had a particular influence on the adaptation of planners with robots in cobotic environments. In 2002, PDDL 2.1 (Fox and Long [2003]) introduced functions to express numerical objects, durative actions and plan-metrics for assessing plan quality. An extension called MAPL (Multi-Agent Planning Language) (Brenner [2003]), introduced finite domain state-variables, actions with duration determined in runtime, and explicit plan synchronisation obtained through speech acts and communications among agents. The newly introduced functionalities are particularly use-

ful in cobotic environments to incorporate actions of varying durations into the planning system, or to allow multi-agent planning for several robots to collaborate in the same environment. In the following years PPDDL (Probabilistic PDDL)(Younes and Littman [2004]) was proposed, extending PDDL 2.1 with probabilistic effects and introducing partial observability. This extension opened up the possibility to integrate statistical models into the planning system and improved the robot’s learning capabilities and performance in uncertain situations. Cobots can benefit from this as they are often faced with uncertain situations when working with different human operators, who have varying behaviours.

3.4 Planners

Task planners have found increasing applications in various areas from aeronautics and space (Aarup et al. [1992]) to agricultural and industrial domains (Robotic Industries Association [2017]). The majority of planning procedures are search procedures, where the difference lies in the search spaces. We conclude this section by introducing the most common techniques for classical planning (Table 3.3): state-space planning, plan-space planning, and planning graphs. We refer the reader to Nau [2007] who provide a more detailed overview of the techniques with accompanying illustrations.

The simplest planners are state-space planners (Ghallab et al. [2004]), which rely on search algorithms in which the search space is a subset of the state space. Each node corresponds to a world state, each arc represents a state transition, *i.e.*, an action, and a plan is a path from the initial state to a goal state in the search space. An extended version of the planner is the Heuristics Search Planner (Bonet and Geffner [2001]), which includes several basic search algorithms like breadth-first search, best-first search (*e.g.*, A*) and iterative deepening search. The Fast Forward planning system (HSP) (Hoffmann and Nebel [2001]) is based on the HSP, but managed to outperform it, due to modifications aimed to avoid getting trapped in local minima. Fast Downward (Helmert [2006]) is a state-space planner based on a finite domain representation, with a greedy best-first search as its main search procedure. A more recent approach is BFWS, a best-first width search (Lipovetzky and Geffner [2017]) that combines width and simple goal-directed heuristic search. The problem with state-space search is that it commits to plan step orderings, meaning that many different orderings of the same actions are considered, even if none of them lead to a solution.

A second category known as plan-space planners, first proposed in TWEAK (Chapman [1987]), handles goal orderings in an optimal way. For such partial-order planners each node is a set of partially instantiated-operators with constraints (*e.g.*, temporal constraints for the application of an action). The idea is to perform a backward search from the goal and impose an increasing number of constraints until a complete plan has been found. Besides later refinements of the initially proposed technique (SNLP (McAllester and Rosenblatt [1991]), UCPOP

Penberthy et al. [1992], O-Plan (Tate et al. [1994])), a more recent system is FAPE, a Flexible Acting and Planning Environment proposed by Dvorak et al. [2014a]. FAPE focuses on integrating acting, task decomposition and temporal planning embedded into a robotic platform.

Another category are planning-graph planners, first introduced in Graphplan (Blum and Furst [1997]). The idea is to encode the search space in a structure called the *planning graph* that is much smaller than the state transition graph and contains information about possibly reachable states. The true set of reachable states can be estimated quickly by eliminating many impossible ones in the search process. A plan is then extracted from the planning graph. Other graph-based planners include SATPlan that uses a SAT solver (Kautz and Selman [1999], Kautz et al. [2006], Rintanen [2014]), and the GP-CSP that uses a CSP solver (Cooper et al. [2011], Do and Kambhampati [2001], Lopez and Bacchus [2003]).

There exist other planning techniques such as model checking (Triantafillou et al. [2015]) or Markov Decision Process (Kolobov [2012]) but there is no single planning strategy or domain-independent search heuristic that is universally better than others.

Table 3.3: Main automated planning approaches and systems

Planner	Generated Plan	Planning Systems
State-space	sequence of actions	HSP (Bonet and Geffner [2001])
		FF (Hoffmann and Nebel [2001])
		FD (Helmert [2006])
		LAMA (Richter and Westphal [2010])
		BFWS (Lipovetzky and Geffner [2017])
Plan-space	partially ordered set of actions	TWEAK (Chapman [1987])
		SNLP (McAllester and Rosenblatt [1991])
		UCPOP (Penberthy et al. [1992])
		O-Plan (Tate et al. [1994])
		FAPE (Dvorak et al. [2014b])
Planning graph	sequence of sets of parallel actions	Graphplan (Blum and Furst [1997])
		STAN (Long and Fox [1999])
		CSP (Lopez and Bacchus [2003])
		weighted CSP (Cooper et al. [2011])
		SATPlan (Rintanen [2014])

The main open-source planning toolkits are the Lightweight Automated Planning ToolKit (Ramirez et al. [2015]), based on Fast Downward and Fast Forward, and PDDL4J, a Planning Domain Description Library for Java cross-platform developers (Pellier and Fiorino [2018]). Integrating classical planning with the Robot Operating System (ROS) has been addressed

recently *e.g.*, with the ROSPlan framework (Cashmore and Fox [2015]) and other libraries such as the ROS package *pddl_planner* (Ueda [2018]) and ROS implementations of planners *e.g.*, PDDL4Jrosy (github.com/pellierd [2017]) and Downward (github.com/phuicy [2014]).

3.5 Knowledge Engineering Tools

Defining planning domain models from scratch can be expensive and error-prone, even for Automated Planning experts. Knowledge engineering (KE) tools for automated planning facilitate this process by automatically encoding a domain model from provided input data. These tools provide support with consistency checks, syntactic error checking, domain visualisation and other functionalities. There exist tools such as GIPO (Simpson et al. [2007]), itSIMPLE (Vaquero et al. [2013]), Planning.Domains (Muise [2016]), as well as plan editors *e.g.*, PDDL Studio (Plch et al. [2012]), and plan visualisers *e.g.*, iGantt (Barták and Skalický [2009]), VIZ (Vodrázka and Chrpá [2010]), VisPlan (Glinský and Barták [2011]), TransportEditor (Škopek and Barták [2017]). Different systems require different inputs such as plan traces, a partial domain model, predicates, or noisy plans. For example, LOCM (Learning Object Centred Models) (Cresswell et al. [2013]) learns action schema from planning traces in Prolog, while itSIMPLE (Fig. 3.5) takes an input in the form of UML (Unified Modeling Language) and generates representations in Petri-Nets and PDDL. Kootbally et al. [2015] created the OWL2PDDL and SQL2PDDL tools to generate PDDL files from OWL files that are stored in a MySQL database for replanning in case of plan execution failures. GIPO (Fig. 3.6) uses Opmaker2 (McCluskey et al. [2009]), a knowledge acquisition and formulation tool that generates a set of PDDL action schema from a given partial domain model and training sequence.

Shah et al. [2013] and Jilani et al. [2014] compare a subset of state-of-the-art KE tools by their learning efficiency, required user experience, system availability and support. Both concluded that most state-of-the-art tools require PDDL experts, or common knowledge in software engineering. However, KE tools that are available online and provide documentation to facilitate the use for beginners remain sparse (*e.g.*, itSIMPLE, GIPO). As most users with different research backgrounds do not have the required background knowledge or expertise, they cannot fully exploit KE tools and task planners in general. This bottleneck significantly reduces the number of potential users in this domain.

3.6 Discussions and Relations to Present Work

Automated planning has found its application in a wide range of areas from virtual agent games (Fernández et al. [2006]) to space exploration (Backes et al. [2004], Bresina et al. [2005]). A main research area has focused on integrating task planning with robot architectures (Cashmore and Fox [2015]), in particular to handle time, resources and synchronisation (Di Rocco

et al. [2014], Dvorak et al. [2014a]). PDDL planners can be very powerful for generating optimal solutions for problems that even humans struggle to solve, *e.g.*, Tower of Hanoi problem (github.com/Elucidation [2012]), Tetris (Vallati et al. [2014]). However, they are generally not accessible to end-users who do not have experience in this domain. While knowledge engineering tools facilitate the modelling of planning domains, they still require PDDL experts or common knowledge in software engineering.

In the previous Chapter 2, we discussed several end-user robot programming techniques and concluded that Programming by Demonstration (PbD) was the most appropriate for our approach. To integrate the taught actions with task planning, the robot needs to not only learn the low-level actions but also their high-level representations in terms of preconditions and effects. Combining task and motion planning is an open research problem and has been addressed previously (Ferrer-Mestres et al. [2015], Garrett et al. [2015]), especially learning preconditions and effects of actions to be used in planning (Ahmadzadeh et al. [2015], Jetchev et al. [2013], Konidaris et al. [2018], Ugur and Piater [2015]). In these works, the focus often lies in grounding actions or learning symbolic action representations, so the robot is generally provided with a predefined set of primitive actions that are refined by learning from plan traces or demonstrations. For example, Abdo et al. [2013] learns a fixed set of manipulation actions from user demonstrations, *i.e.*, stack blocks, pour from a bottle and open a door. The robot only uses the actions to achieve the predefined task, but does not reuse them for new tasks. In this thesis we address scenarios where the robot has no predefined set of actions and learns them from scratch, in particular, from human demonstrations.

In the first part of the thesis, we reviewed state-of-the-art techniques in end-user robot programming and automated planning. We identified Programming by Demonstration as an appropriate approach for end-users to teach low-level actions from scratch. However, they also need to teach the high-level action representation in terms of preconditions and effects to allow the use of planners. Task planning has been integrated in robotics systems, but they are generally not accessible to end-users without experience in Automated Planning or related fields. In this thesis, we argue that these end-users can still learn and use these concepts to teach robots reusable actions to solve problems with task planners. In the second part of this thesis we present our contributions, where we propose an end-user robot programming framework that combines PbD and Automated Planning.

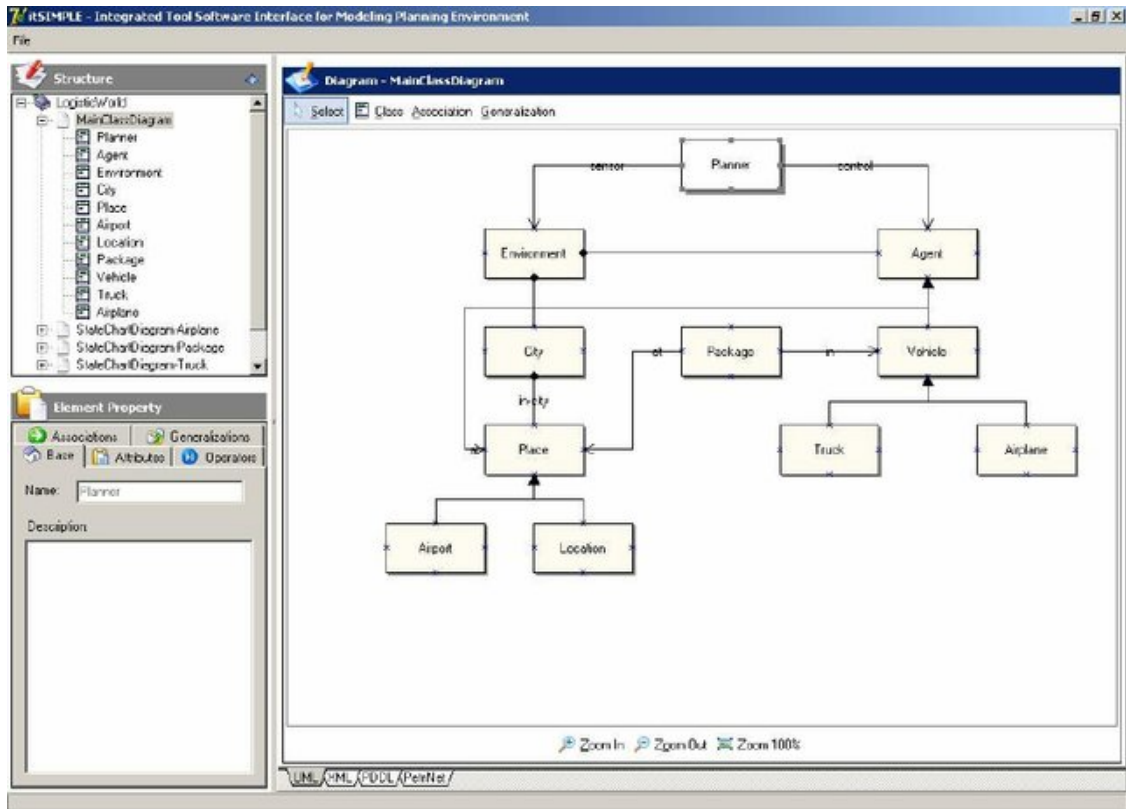


Figure 3.5: Screenshot of a Planning UML model in itSIMPLE software (Vaquero et al. [2013])

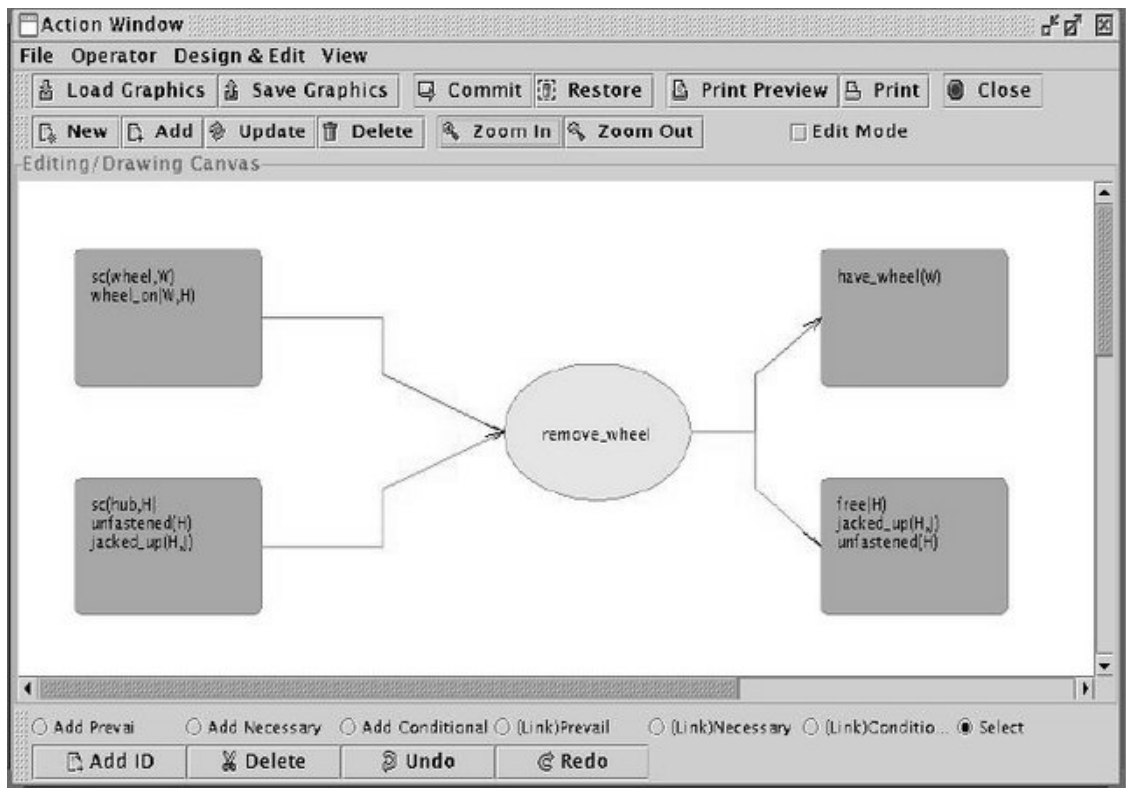


Figure 3.6: Screenshot of the operator editor tool in GIPO (McCluskey and Simpson [2005])

Part II

Contributions

Chapter 4

An End-User Robot Programming Framework for Cobotic Environments

Contents

4.1	iRoPro - interactive Robot Programming	56
4.1.1	Programming by Demonstration: teaching actions	57
4.1.2	Automated Planning: reusing actions	58
4.1.3	Retro-active Loop: refining actions	58
4.2	Methodology	59

Programming robots for general purpose applications is extremely challenging due to the great diversity of end-user tasks. Not only is it generally left up to robotics experts, but end-user programming solutions are often limited to teaching robots predefined actions for specific tasks that cannot be reused. This thesis argues for letting end-users teach robots primitive or *atomic* actions from scratch that can be reused with task planners for previously unseen problems. In this way, we delegate the logical reasoning process of finding a solution to task planners and facilitate the robot programming process, while maintaining the generalisability of taught actions.

This part of the thesis contains our contributions. In this chapter we present iRoPro, an interactive Robot Programming framework and discuss its theoretic components. The framework combines Programming by Demonstration and Automated Planning techniques for goal-oriented robot programming by end-users. We end this chapter by discussing the research methodology used in this thesis which resulted in the contributions presented in the remaining chapters (Sec. 4.2).

4.1 iRoPro - interactive Robot Programming

iRoPro, an interactive Robot Programming, is a framework that allows end-users to teach robots primitive actions from scratch that can be reused with task planners. The framework consists of the following three components (Fig. 4.1):

- A. Programming by Demonstration: The user *teaches* the robot primitive actions by demonstration. The robot creates an action model that the user can refine and validate.
- B. Automated Planning: The user defines a new planning problem with a goal to achieve. The robot which *reuses* the taught actions with a planner to generate a 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 graphical user interface (GUI) that abstracts from the underlying modeling 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 discuss each component in more detail.

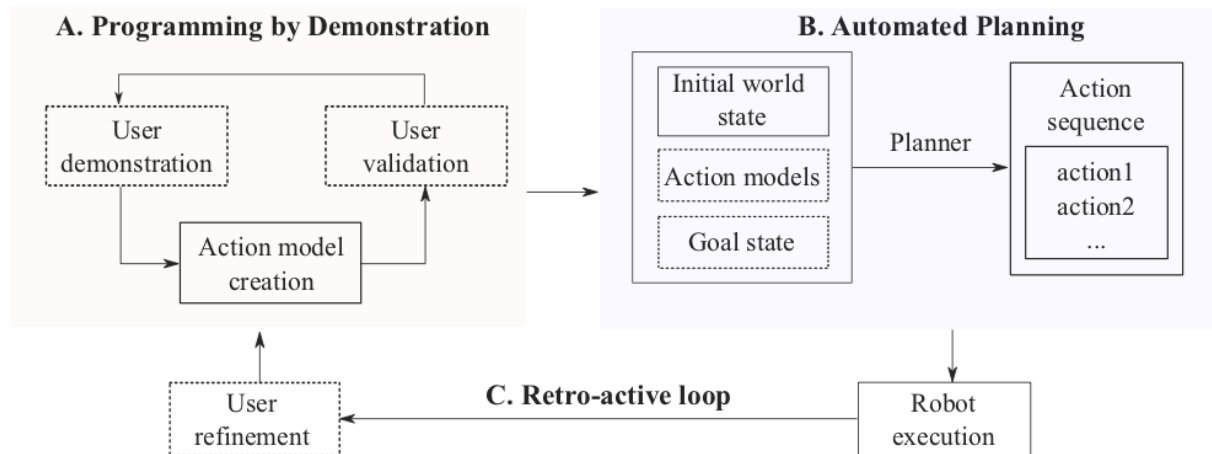


Figure 4.1: An overview of the iRoPro (interactive Robot Programming) framework: A. the user teaches primitive actions by demonstration B. the robot reuses these with a task planner to generate an action sequence to achieve a goal. C. After observing the robot execution, the user can refine the taught action models (dotted lines indicate user actions, solid lines indicate robot actions).

4.1.1 Programming by Demonstration: teaching actions

Teaching robots atomic actions consists of learning both *how* and *when* an action should be applied, *i.e.*, the low-level (Sec. 2.3.1.1) and high-level representations (Sec. 2.3.1.2) respectively. We consider an action that consists of both low- and high-level representations an *action model*. To teach action models, Programming by demonstration (PbD) (Sec. 2.3.4) can be used as an intuitive end-users programming approach. Low-level actions can be learned from multiple demonstrations (Niekum et al. [2012]) or a single demonstration, where poses are assigned heuristically and corrected by the user if needed (Alexandrova et al. [2014]). Dynamic Movement Primitives (Pastor et al. [2009]) or mixture models (Calinon and Billard [2007b]) learn actions from entire motion trajectories but generally require multiple demonstrations (Abdo et al. [2013]). Ahmadzadeh et al. [2013] generates trajectories by extracting three key points that represent the *rest*, *pick* and *place* actions of an object. In keyframe-based PbD (Akgun et al. [2012], Alexandrova et al. [2014]), actions are represented as a sparse sequence of keyframes that can be connected to perform a skill.

The user can teach multiple low-level actions and discriminate between them by associating different high-level conditions that specify *when* the robot should use the action. In iRoPro, high-level actions are represented similar to planning operators in automated planning (Sec. 3.2), where an action is a tuple $o = (\text{name}(o), \text{precond}(o), \text{effect}(o))$ with preconditions and effects. State-of-the-art perception systems (*e.g.*, SIFT (Ahmadzadeh et al. [2015])) or a database of object features (Mason and Lopes [2011])) are used to automatically recognise object properties such as type, position, or spatial relation relative to other objects. When the user demonstrates an action, such as pick-and-place of a cube, it results in a change in the world state, *e.g.*, the cube’s position changes from A to B. The robot observes the world state before and after the action demonstration, and infers relevant predicates for preconditions and effects to build an action model. Predicates can be inferred from observing what changed or what stayed the same in the state of the world. Feature-based algorithms, such as k-means clustering, can be used to learn high-level action conditions from multiple demonstrations (Abdo et al. [2013], Mollard et al. [2015]). Existing PbD approaches try to teach the robot from a small number of demonstrations, but require at least five contextually different ones (Abdo et al. [2013], Orendt et al. [2016]). In iRoPro, we propose an interactive programming approach, where the user can directly modify learned action models via the graphical interface. Relying on the user’s logical reasoning and understanding of what they want to teach the robot, we allow them to directly program and correct 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 unique primitive action.

4.1.2 Automated Planning: reusing actions

In the PbD step, the robot learned action models that include the high-level representation that is used in Automated Planning. The Automated Planning step consists of creating a planning problem, defining a task goal and generating a solution to solve the problem. Task planners are used to generate solutions to solve complex problems (Chapter 3). Various planners have been used for robotic task planning, such as the STRIPS planner (She et al. [2014]), Metric-FF (Cubek et al. [2015]), fast downward planner (Abdo et al. [2013]). Given a description of a planning *domain*, *i.e.*, object types, predicates, and actions, we can define a planning *problem* with an initial state and a desired goal state to achieve. The planner generates an optimal action sequence, or *plan*, which guarantees the transition from initial state to the goal state.

Depending on the robot architecture and perception system, iRoPro integrates a partial PDDL domain including a set of object types and their predicates that the robot can recognise. The previously created action models complete the partial PDDL domain. Using the graphical interface, the user can create a new planning problem by detecting the initial world state and defining a goal to achieve. The predicates for the initial world state are automatically inferred by the robot as in the PbD phase. The user can modify and correct them via the interface. Then they enter predicates that describe a goal for the robot to achieve. 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 actions, as well as the initial and goal states and relaunch the planner.

4.1.3 Retro-active Loop: refining actions

The retro-active loop allows the user to revisit and correct created action models. It is likely that the initially generated plan does not produce the desired outcome, especially if the context of the planning problem is different to that of the initial demonstration (*e.g.*, different object types or positions). To minimise the user's programming process and the number of demonstrations required, taught actions can be generalised and reused, especially if the low-level action is similar. Instead of creating new action models for each new problem, the user can revisit and modify existing ones so that they are reused by the planner. Thus, the application to a new context is an important step to test the generalisability of action models. There are several possible causes for suboptimal, incorrect or non-existent solutions generated by the planner:

- **Object types:** they dictate what objects an action can be applied to. If they do not match those of the observed world state in the current planning problem, the action is not considered by the planner (*e.g.*, pick-and-place was only defined for cube objects but not other types).

- **Preconditions:** they define when an action can be applied. If they do not match the observed world state, the planner could make wrong assumptions about the usage of taught actions (*e.g.*, pick-and-place of an object that is not 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 perceived world state and the actual world state (*e.g.*, a position is still considered free when it is occupied).
- **Initial world states:** they describe the existing world state of objects to the robot. If the initial world state is incorrect the planner may consider certain actions as invalid due to their defined preconditions.
- **Goal:** this defines the minimal set of predicates that need to be achieved and should not include consequent states or intermediate steps to achieve the goal. Contradicting goal states automatically lead to non-existing plans (*e.g.*, 'object is on A' and 'A is clear' are stated as goal states).

Knowledge engineering tools (Sec. 3.5) can facilitate this process of modifying action models. They often provide useful functionalities for dynamic testing, model checking and visualisation (Simpson et al. [2007]), but most tools require expertise in automated planning or Software Engineering. In this thesis we argue that the proposed robot programming process does not require this expertise and can be learned easily by users with different educational backgrounds.

4.2 Methodology

The contributions in this thesis were constructed using both quantitative and qualitative research methodologies. Our general approach follows the design wheel of the concept design process (University of Cambridge [2013]) consisting of successive cycles of 'Explore', 'Create', 'Evaluate' and 'Manage' phases (Fig. 4.2). Our contributions represent the following stages:

1. **Pre-Experiments:** We start by conducting initial qualitative user experiments to investigate how end-users adopt basic concepts in Automated Planning and Programming by Demonstration. We are particularly interested in the difficulties they encounter when learning and applying automated planning concepts as they can be considered in the iRoPro system implementation. For this we create an initial prototype used for simulating iRoPro with the Wizard-of-Oz technique (Chapter 5).

2. **Goal-oriented Programming:** We then explore goal-oriented end-user programming approaches by allowing users to simultaneously teach the robot actions and goals by demonstration. For this we implement a system that includes learning new actions by demonstration and a graphical interface. We evaluate the system in an online user study on Amazon Mechanical Turk (Chapter 6).
3. **End-to-end System Implementation:** Taking the existing work as a basis, we implement iRoPro on a Baxter robot to allow simultaneous teaching of low- and high-level actions by demonstration. The end-to-end system includes a graphical interface that users interact with directly to teach, reuse and refine new action models (Chapter 7).
4. **Post-Experiments:** We conclude this thesis work by conducting further user experiments using the implemented system. We compare user groups with different educational backgrounds and investigate their performance on how they learn and use the system for programming the robot. We close the concept design process by comparing the latest user study with our initial experiments conducted at the start of this thesis work (Sec. 7.5).

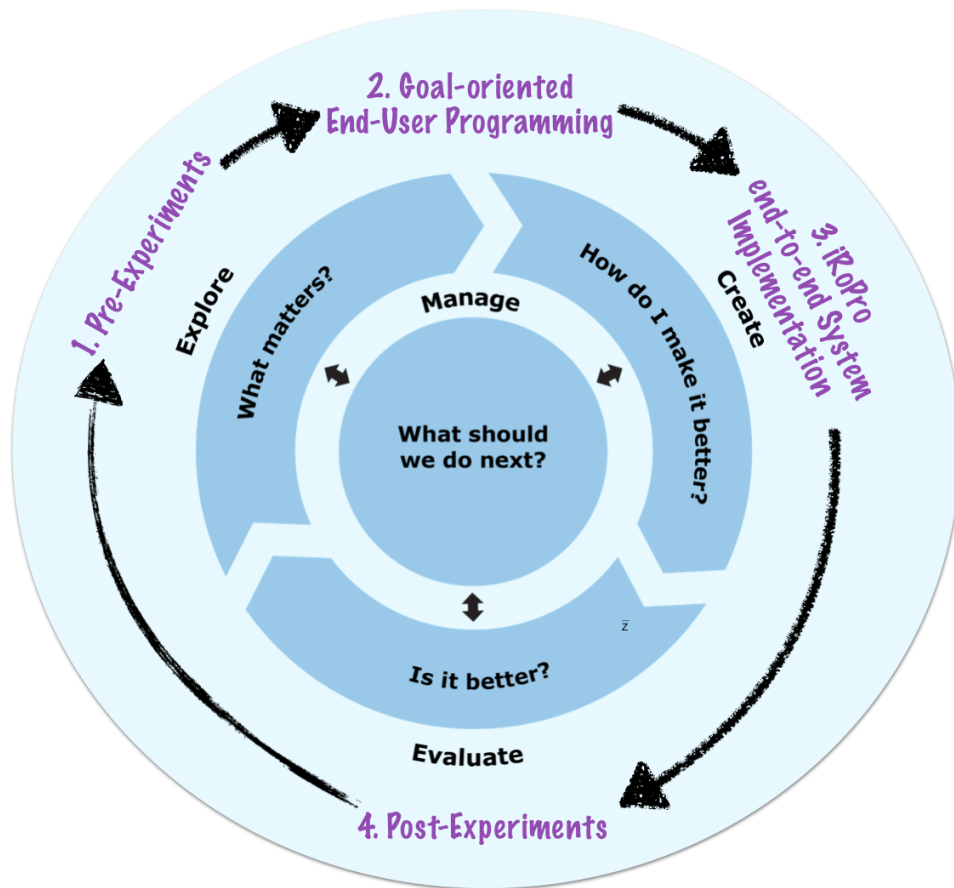


Figure 4.2: The thesis contributions align with the design wheel consisting of successive cycles of ‘Explore’, ‘Create’, ‘Evaluate’ and ‘Manage’ phases (University of Cambridge [2013]).

Chapter 5

Pre-Experiments

Contents

5.1	Baxter Research Robot	62
5.2	Experiment 1: Acceptance of Automated Planning and PDDL Concepts .	62
5.2.1	Experimental Setup & Participants	63
5.2.2	Experimental Design & Measurements	64
5.2.3	Results	66
5.3	Experiment 2: Acceptance of the Robot Programming Framework	67
5.3.1	Experimental Setup & Participants	67
5.3.2	Experimental Design & Measurements	68
5.3.3	Results	69
5.4	Findings	70
5.5	Conclusions	71

In the previous chapter (Chapter 4) we proposed iRoPro, an interactive Robot Programming framework that allows end-users to teach robots new actions by demonstration and can be reused with task planners to complete previously unseen tasks. We believe that non-robotics expert users with little to no experience in Automated Planning can easily learn to use this framework. In this chapter we present two qualitative user experiments to respond to the following questions:

- Q1** How do non-expert users adopt the automated planning language with its action model representation? (Section 5.2)
- Q2** Can users teach a robot action models for automated planning using the proposed iRoPro framework? (Section 5.3)

The experimental context was designed around a Baxter robot (Fig. 5.1a). 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. In the following sections we briefly outline the experimental setup, protocol (consisting of training, experimental test, planning, questionnaire and debriefing), measurements and results for each experiment. We conclude this chapter with a discussion on the experimental findings and notable aspects for a full implementation of the framework.

5.1 Baxter Research Robot

In this thesis we worked with a Baxter Research Robot, created by Rethink Robotics [2016]. First released in 2012, Baxter is a two-armed humanoid robot, with 7-DoF and a load capacity of 2.2kg each. Our version has one claw and one suction gripper (Fig. 5.1a). The SDK interfaces with Baxter via the Robot Operating System (ROS) (Quigley et al.), a framework developed in 2007 by the Stanford Artificial Intelligence Laboratory, that allows the shared use of software across a wide variety of robotic platforms (Fernández et al. [2015]). The industrial model of the Baxter robot comes with the Intra software (Rethink Robotics [2012]) that provides a graphical user interface, allowing recording and replaying of joint trajectories. The research model of the robot comes without the software but with a head-mounted camera and sonar head sensors. Several research laboratories have developed algorithms using the Baxter robot to implement state-of-the-art solutions *e.g.*, to pick up golf balls and place them in a basket (Vyvyan Pugh [2014]) or to play the game Connect 4 by picking up chips from a specified location (Rethink Robotics [2014]). Recent approaches have used Baxter with various PbD techniques (Li et al. [2017], Tremblay et al. [2018a], Yang et al. [2016]). After the Ebola outbreak in West Africa in 2014, Baxter has been used to reduce the risk of contamination (Active8 Robots [2014]). In October 2018, Rethink Robotics closed down and was acquired by German automation specialist HAHN Group (Crowe [2018]).

5.2 Experiment 1: Acceptance of Automated Planning and PDDL Concepts

In this experiment, we address the following question:

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

Users were introduced to a symbolic planning language (a simplified version of PDDL),

involving the STRIPS formalism (Fikes and Nilsson [1971]) with type structures used in automated planning (Chapter 3). Users were instructed to describe world state configurations to the robot. The goal was to assess the user’s adoption of the planning concepts (*i.e.*, object types, properties, generalised properties, action models) and to verify that the symbolic planning language is appropriate for non-expert users.

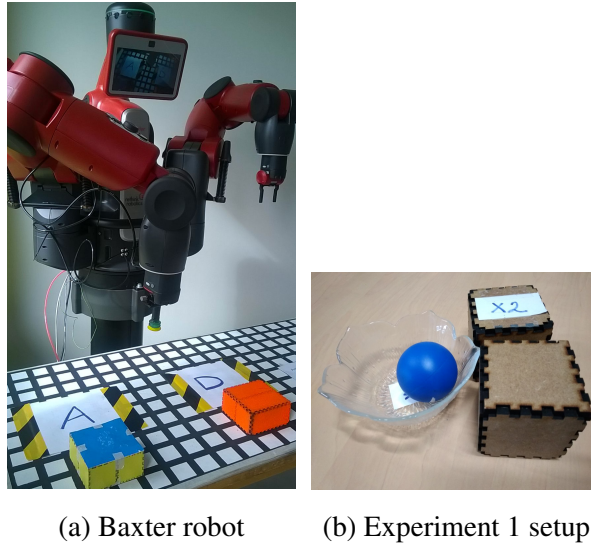


Figure 5.1: Experimental setup for user studies.

5.2.1 Experimental Setup & Participants

We recruited 10 participants (1 male, 9 female), who were sociology students at the Université Grenoble Alpes. 3 participants reported no programming experience, 6 had experience with office productivity software (‘beginner’), and 1 had previously taken a programming course before (‘advanced’).

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 (Fig. 5.1b). 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 experiment was 49 minutes. At the end, participants were given a questionnaire related to their experience and their understanding of the learned planning language and concepts. The participants’ behaviour was observed by the experimenter and the experiment was recorded on camera. The experimental protocol, questionnaire and additional material used can be found in Appendix B.

5.2.2 Experimental Design & Measurements

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. At the course of the experiment, users were faced with three different scenarios of increasing complexity. We evaluated their capability to learn the presented planning language and apply it to different problem statements. The experiment consisted of the following phases:

- **Training:** Users were presented the symbolic planning language to describe object *types* (i.e., position, ball, cube, bowl) and predicates, which we called *properties* (i.e., empty, at, stackable, is_red, is_blue) to describe world states. They were shown how to model a simple move action in terms of preconditions and effects (Fig. 5.2). For all properties, actions and their parameters, they had to use syntax of the form `name(arg1, arg2, ...)` which users without a Computer Science background might be unfamiliar with. Additionally, they were introduced to the concepts of *instantiated* and *generalised* actions, which were equivalent to actions (e.g., `move(X1)`) and planning operators (e.g., `move(cube)`) respectively (Sec. 3.2). In this phase, they were given a simple example of a cube at position A1 and moved to position B2.

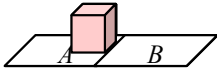
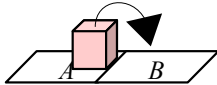
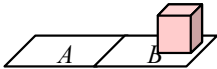
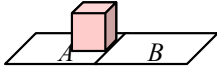
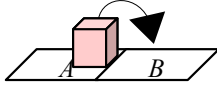
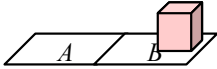
Preconditions: (at cube A) (empty B) 	(move cube A B) 	Effects: (at cube B) (empty A) 
Preconditions: (at ?obj ?posA) (empty ?posB) 	(move ?obj ?posA ?posB) 	Effects: (at ?obj ?posB) (empty ?posA) 

Figure 5.2: Action model representation of a move action in terms of preconditions and effects: an action (or instantiated operator) for a cube (top), and generalised action (or planning operator) for any object, where variables are prefixed with ‘?’ (bottom).

- **Experimental test:** Users were presented a new world state that involved a cube, a bowl and a ball object. First they were instructed to provide a description of the initial state to the robot by using the symbolic planning language (Fig. 5.3a). Then they were asked to define a move action model in terms of preconditions and effects (Fig. 5.3b). In the following they were faced with 3 different scenarios to refine the preconditions of the

move action. Users derived a (*stackable ball cube*) property (Fig. 5.3c), which allowed a ball to be stacked on top of a cube. When this property did not hold, users proposed the *empty* property (Fig. 5.3d), which the robot needed to verify before the action execution. At each step, users had to give the generalised representation of the properties and action models.

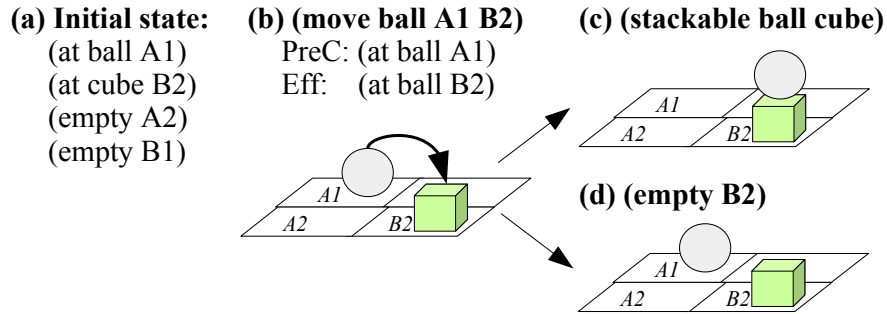


Figure 5.3: Users were instructed to provide a description of (a) the initial state of the world and (b) an initial move action model. Then they derived additional preconditions for moving the ball from position A1 to B2: (c) (*stackable ball cube*): the ball can be stacked onto the cube, and (d) (*empty B2*): if the ball cannot be stacked, the target position should be empty.

- **Planning:** Users were presented a description of a new initial state of the world and a goal. They were asked to define an action sequence that allows the transition from the initial state to the goal (similar to Fig. 5.4), and explain their reasoning using the symbolic action model representation. This optional test allowed us to further verify their understanding of the planning concepts, in particular action preconditions and effects.

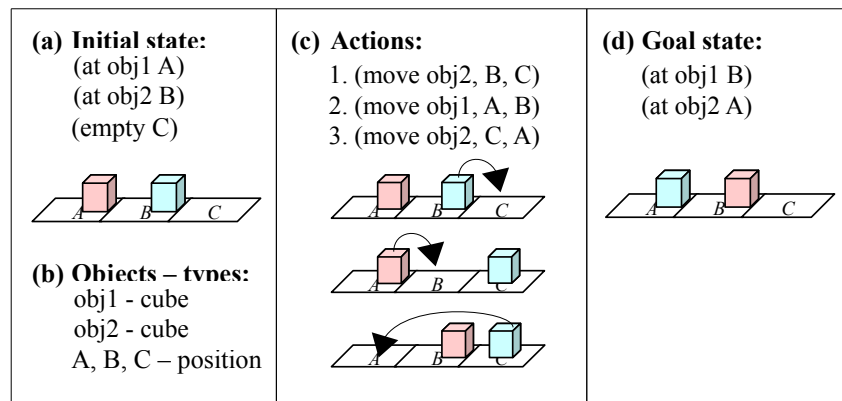


Figure 5.4: Definition of a planning problem (a) properties describing the initial world state (b) object names and their types (c) instantiated actions (d) properties describing the goal.

- **Questionnaire:** At the end, users were given a questionnaire with 5 questions related to their experience and 15 questions to evaluate their understanding of the learned planning

language and concepts. For the latter, questions related to their understanding of the concepts presented at the start of the experiment (*e.g.*, ‘Explain the difference between the precondition and effect of an action’), syntax (*e.g.*, ‘Using the presented language, how do you describe the property *cube X4 is on position B3?*’), logical reasoning (*e.g.*, ‘Is it possible to have (empty A) and (at cube A) in the same state?’), and other concepts (*e.g.*, ‘What is the generalised form of the given object property?’).

- **Debriefing:** Throughout the experiment, users were asked open-ended questions (*e.g.*, ‘What properties do you observe in the current world state?’), so that they were guided as little as possible and their responses were unbiased. When the participant struggled to find an answer, the experimenter guided the participant in a possible direction (*e.g.*, ‘Why can the ball not be placed on the cube?’).

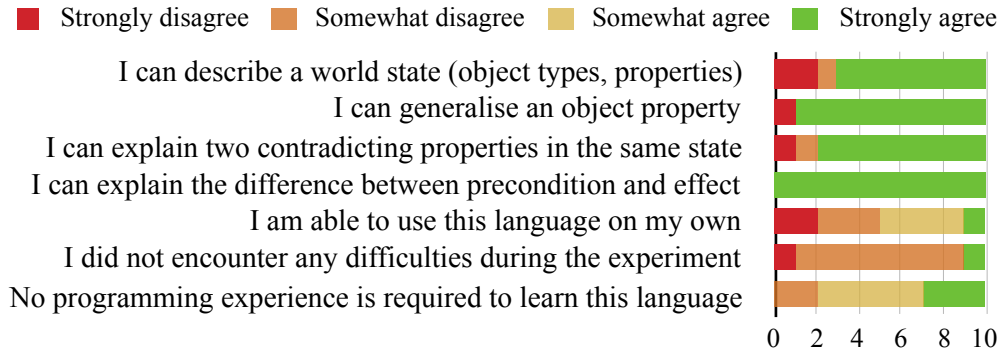


Figure 5.5: Extract of questionnaire responses from Experiment 1.

5.2.3 Results

We did not observe any significant differences in the performance of users with or without programming experience. 9 (out of 10) participants found the symbolic representation of properties and actions easy to understand. During the experimental test, the majority (9 or 90%) of the participants managed to describe the complete world state using the correct syntax. When faced with different scenarios to refine the action model, 5 (or 50%) of the participants struggled to formalise the *stackable* condition in the symbolic language. They provided alternative formulations related to the cube’s properties (*e.g.*, ‘if the cube can hold the ball’). However, once the condition was defined, the majority (8 or 80%) of the participants had little to no difficulties defining the planning operators (Fig. 5.2). Due to time constraints, only 5 (or 50%) participants were presented the planning phase. All 5 encountered no problems when defining the action sequence to achieve the given goal.

In the questionnaire (Fig. 5.5), the majority (9 or 90%) of the participants understood the notion of states and object properties. 8 (or 80%) correctly pointed out two properties that could not exist in the same state (*e.g.*, (empty A) and (at cube A)). All participants

gave correct explanations for preconditions and effects of action models, and provided correct examples. 9 (or 90%) participants encountered difficulties during the experiment, 6 (or 60%) stated problems with formalising the language, especially at the beginning of the experiment. Half of the participants believed that they could apply this language on their own. 7 (or 70%) stated a ‘beginner’ programming level was needed to learn the symbolic planning language, while 3 (or 30%) believed that no programming experience was required at all.

5.3 Experiment 2: Acceptance of the Robot Programming Framework

In this experiment, we address the following question:

Q2 Can users teach a robot action models for automated planning using the proposed framework iRoPro?

Users were presented a simulated implementation of iRoPro and had to teach action models by kinesthetically manipulating a Baxter robot (Fig. 5.1a). Users were instructed to teach an atomic 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.

5.3.1 Experimental Setup & Participants

We recruited 11 participants (7 male, 4 female), who were students and staff members at the Université Grenoble Alpes¹. 6 participants reported programming experience with office productivity software (‘beginner’), 2 had previously taken a programming course before (‘advanced’), and 3 were pursuing studies in Computer Science (‘expert’). The experiments were conducted using a Baxter robot, mounted with a partial implementation of the framework. The implemented functionalities included:

- ‘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.

¹None of the participants took part in the first experiment

We used the Wizard-of-Oz technique to simulate the remaining functionalities (*e.g.*, ‘infer action preconditions and effects’, ‘generate solution using a planner’). Participants operated on a table with 2 positions D (for departure) and A (for arrival), 2 cubes (blue and red), that represented parts on an assembly line (Fig. 5.1a). 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. The experimental protocol, questionnaire and additional material used can be found in Appendix C.

5.3.2 Experimental Design & Measurements

The experiment scenario was set in a simulated assembly line, where objects of the same shape, but different colour arrived consecutively at the departure position D. Users were told that objects were too heavy for human operators to move, hence needed to be handled by robots. Due to the type of the objects, they should not be stacked either. Users had to teach Baxter the action for moving an object from D to arrival position A, where another maintenance task would be performed later. At the course of the experiment, users were faced with two different scenarios, where Baxter had to apply the learned move action. We evaluated the user’s capability to refine action models and associate conditions when faced with different situations, and assessed the framework’s overall usability. The experiment consisted of the following phases:

- **Training:** Users were shown how to manipulate Baxter’s arm to pick and place an object, and given time to familiarise themselves with the kinesthetic manipulation. For this experiment we only used the robot’s suction gripper to manipulate objects.
- **Experimental test:** Users were instructed to teach Baxter a move action of a red cube. Then, they were presented the action model, with preconditions and effects, that Baxter learned from the demonstration (Fig. 5.6a). In the following, users were faced with two different scenarios to refine the conditions of the action model, starting with the initial action model for a red cube. At each step, users observed how Baxter executed the learned action in the new scenario. When Baxter failed to execute the action, users had to refine the conditions of the action model so that it was applicable to all cubes of any colour (Fig. 5.6b) and when the target position was occupied (Fig. 5.6c).
- **Planning:** Users were presented a new scenario, where Baxter was instructed to achieve a goal using the learned action model. The new goal was to switch the positions of two cubes on the table. Users were first asked if they believed Baxter was able to solve this task and were then shown how the taught action was reused with a task planner. Finally, Baxter executed the action sequence to complete the task (Fig. 5.4).
- **Questionnaire:** At the end, users were given a questionnaire containing 18 questions related to their experience (*e.g.*, ‘I did not encounter any difficulties during the experi-

ment’), their perceived understanding of the presented concepts (*e.g.*, ‘I can explain how Baxter represented the preconditions of a new action’), and the usability of the framework (*e.g.*, ‘No programming experience is required to teach Baxter a new task’). Participants had to give a rating on a 4-point scale ranging from ‘Strongly agree’, ‘Somewhat agree’, ‘Somewhat disagree’, and ‘Strongly disagree’. The complete questionnaire can be found in Appendix C.

- **Debriefing:** Throughout the experiment, users were asked about their expectations on Baxter’s behaviour before applying the learned action model in a new scenario. Users were asked open-ended questions (*e.g.*, ‘What will Baxter do when applying the learned action model?’), so that their responses were unbiased. When they encountered failure scenarios (*e.g.*, when Baxter stacked two cubes), they were asked to reason about Baxter’s behaviour and proposed modifications to the taught action model.

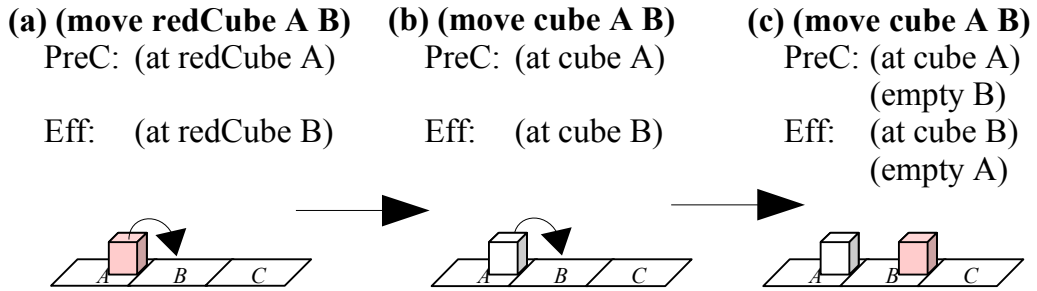


Figure 5.6: Continuous refinement of the move action model: (a) initial action model learned by demonstration, (b) action model for all cubes of any colour, (c) action model with an additional condition, if the target position is occupied and cubes can not be stacked.

5.3.3 Results

During the experiments we observed how users learned and used the presented programming process and planning concepts. When asked for improvements of the initial action model no users pointed out missing conditions before being faced with the new scenario. Even users who were ‘experts’ and who have heard of automated planning before, did not propose a complete action model from the start. However, when faced with the relevant failure scenarios all of the users detected the missing conditions easily. In the final phase, 8 (or 73%) users who had no experience in automated planning did not expect Baxter to solve the permutation problem.

Figure 5.7 shows the user responses to the questionnaire. All 11 users were satisfied with the programming process and Baxter’s ability to learn and reproduce the demonstrated move action. All users stated that they encountered no difficulties during the experiment and believed that they had taught Baxter a new task. The majority of the users agreed that they could explain

how Baxter learned and represented the new action model. 9 (or 82%) understood the notion of preconditions and agreed that no programming experience was required to teach Baxter using the proposed framework, while 2 (or 18%) somewhat disagreed.

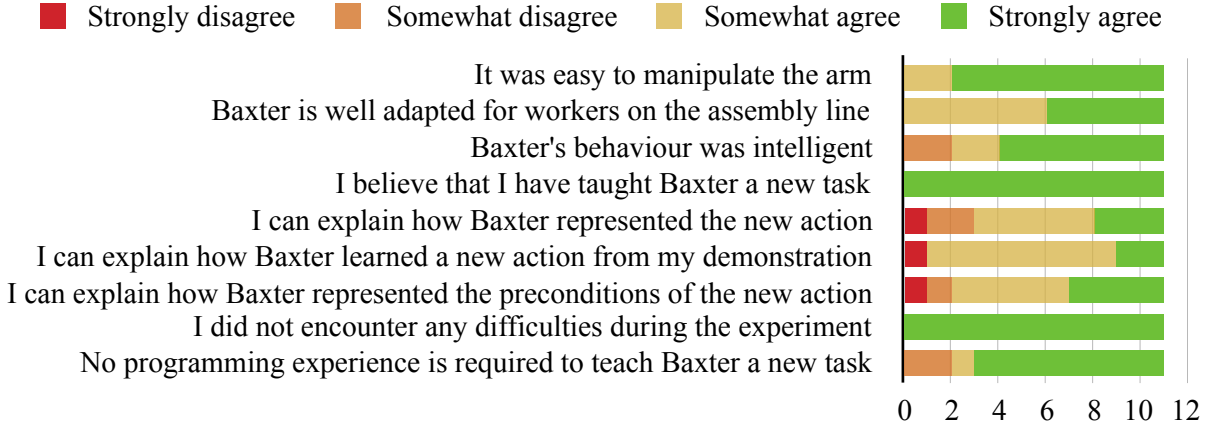


Figure 5.7: Summary of questionnaire responses: Extract of 18 questions on the user’s perceived usability and understanding of the programming process after the experiment.

5.4 Findings

In both experiments, we did not observe a significant difference in the performance between users with different programming experience. The majority of the users had issues formulating the logical properties used for preconditions and effects. In the first experiment (Sec. 5.2), 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 (Sec. 5.3), 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*’). This means that users should be provided with predefined conditions that they can choose from, instead of letting them formulate their own.

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 (Suay et al. [2012]). 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.

5.5 Conclusions

In this chapter we evaluated potential impacts of iRoPro, the robot programming framework proposed in Chapter 4 with qualitative experiments. The framework combines two techniques, Programming by Demonstration and Automated Planning, allowing end-users to teach action models from scratch. Users with no background in Computer Science had initial difficulties remembering the syntax of the symbolic planning language, but managed to use it after less than an hour of training. With both experiments we showed that non-expert users can easily learn the main automated planning concepts, even when introduced to them for the first time. In particular, the action model representations in terms of preconditions and effects seem intuitive for users despite their different educational backgrounds. Overall, the proposed robot programming process was considered to be very intuitive and easily understood by users.

Now that we have verified the possible usability of iRoPro, we need to validate its actual usability with the implementation of a working end-to-end system. This involves using state-of-the-art solutions to implement functionalities that were simulated with the Wizard-of-Oz technique during the experiments. This consists of the following key aspects:

1. learn generalisable low-level actions by demonstration (*e.g.*, Akgun et al. [2012], Pastor et al. [2009]),
2. learn high-level action representations in terms of preconditions and effects (*e.g.*, Abdo et al. [2013], Mollard et al. [2015]),
3. integrate a task planner to generate solutions using the learned actions (*e.g.*, Abdo et al. [2013]),
4. create a graphical user interface to guide the end-user programming process and an intuitive navigation between all functionalities (*e.g.*, Alexandrova et al. [2014], Huang and Cakmak [2017]).

While we have validated the learnability of the main concepts in Automated Planning, we will now focus on the subset of goal-oriented programming. Goal-oriented programming refers

to high-level programming by providing the user with an abstraction from the underlying modeling language, *e.g.*, via a user interface. This allows end-users to understand and learn the programming process, without having to dive into the syntax and the technical details of the modeling language. In the next Chapter (6) we present a goal-oriented programming system that allows end-users to program robots via an intuitive graphical interface. Our approach uses keyframe-based PbD (Akgun et al. [2012]) to teach the robot generalisable low-level actions that can be used to achieve a variety of different goals in the same task domain. This system represents a foundation for the iRoPro implementation.

Chapter 6

Goal-Oriented End-User Robot Programming

Contents

6.1	Use Case	74
6.1.1	Related work	76
6.2	Approach	77
6.2.1	Freiburg Dataset Analysis	77
6.2.2	Shelf Arrangement Representation	79
6.2.3	Goal Inference from Demonstration	80
6.2.4	Goal Inference with Direct Specification	81
6.2.5	Action Representation and Inference	81
6.2.6	Implementation	82
6.3	Goal Inference Evaluation	82
6.3.1	Teaching Strategies	83
6.3.2	Evaluation on the Freiburg Dataset	84
6.3.3	User Study Evaluation	86
6.4	System Evaluation	89
6.4.1	Protocol	90
6.4.2	Results	90
6.5	Discussions	90
6.6	Conclusions	92

In this chapter, we focus on enabling end-users to program robots using a goal-oriented programming approach. We make use of a graphical user interface (GUI) that facilitates the programming process by providing the user with an abstraction from the underlying modelling language. As treating all possible application domains is beyond the scope of this thesis, we will focus on the specific use case of shelf organising tasks, which generally involves basic pick-and-place actions of shelf products. We choose keyframe-based PbD (Akgun et al. [2012]) as the state-of-the-art technique to allow end-users to teach robots low-level actions by demonstration. We implement a system where end-users teach low-level actions that can be customised with the GUI. Both, the presented system and the chosen use case build firm foundations for this thesis and for the implementation of iRoPro, addressing assembly and packaging tasks with pick-and-place robots.

In the following sections we first give an introduction to the problem statement of the chosen use case of robotic shelf organisation and discuss related works (Sec. 6.1). Then, we propose a task representation for shelf arrangements based on a large dataset of grocery store shelf images and a method for inferring goal configurations from user inputs (Sec. 6.2). We evaluate our goal inference approach with ten different teaching strategies that combine alternative user inputs on the dataset of grocery configurations and with real human teachers through an online user study (Sec. 6.3). Finally, we evaluate the robot programming system implemented on a Fetch mobile manipulator on eight benchmark tasks and demonstrate real-time execution of shelf arrangement tasks (Sec. 6.4). We complete this chapter by discussing the limitations of the system (Sec. 6.5) and relations with this thesis (Sec. 6.6).

6.1 Use Case

The supermarkets and grocery stores industry employs millions of workers for tedious tasks, including restocking products on shelves. These tasks have certain regularities that can be exploited by automation solutions. For instance, most objects are rectangular prisms (*e.g.*, boxes) or cylinders (*e.g.*, cans) and they are often organised on a shelf in a grid pattern with the label facing forward. On the other hand, the arrangement task is slightly different for every item, with varying grid configuration parameters (rows, columns, stacks, or object distances) due to differences in shelf space, product types, and product dimensions. Furthermore, different robot end-effectors require different ways of manipulating objects to get them tightly arranged in confined shelf settings. As a result, developing universal robotic shelf arrangement capabilities that work for all possible items in all possible stores is extremely difficult.

Instead, we argue for robot shelf arrangement tasks to be programmed by end-users at the time of deployment. Rather than developing universal capabilities, we embrace the idea that a robot will be customised to a specific store and the specific items in it. While this presents a

simpler programming challenge, enabling end-users to do it is non-trivial. The system needs to not only be intuitive and easy to use, but it should also allow efficient programming of both *what* the desired arrangement of items looks like and *how* the robot can use its manipulator to move each item to the desired position relative to other items.

In this chapter we propose an approach for enabling users to simultaneously program robotic shelf arrangement task goals and actions for a given item, by demonstrating a few steps of the shelf stocking task (Fig. 6.1). We develop a user interface to visualise inferred task goals and actions, and enable other user input to augment their demonstrations. To better understand common structure in shelf arrangement tasks, we analysed the Freiburg dataset of close to 5,000 images covering more than 2,400 unique grocery items from 25 different categories (Jund et al. [2016]). Our goal inference model, based on this dataset, takes demonstrations and other input from the user and proposes them most likely shelf arrangements in order to accelerate the teaching process. We analyse how quickly correct goal configurations in the dataset can be inferred from user input according to 10 different teaching strategies and present an online user study that empirically investigates strategies that people use in a simplified arrangement task domain. We implement our approach on a Fetch mobile manipulator and demonstrate the programming and execution of 8 arrangement tasks for objects from the dataset.

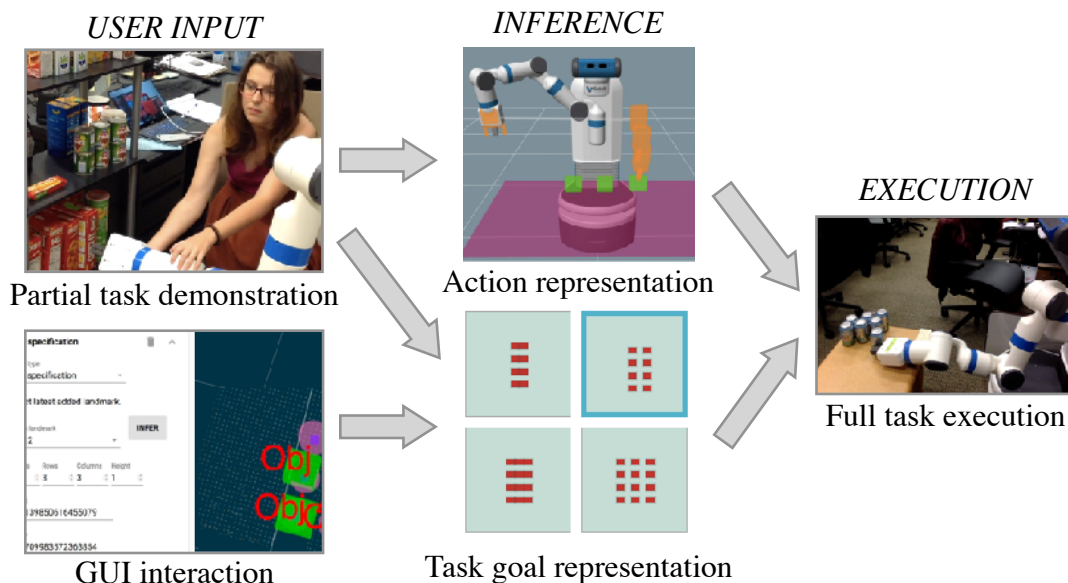


Figure 6.1: Overview of the developed system that allows users to demonstrate part of a shelf arrangement task and interact with a GUI to simultaneously program both the complete task goal (fully specified shelf arrangement) and the actions for achieving that goal.

6.1.1 Related work

This work relates to several topics explored in previous robotics research. The larger umbrella of end-user robot programming has seen a range of recent work focused on programming of mobile robots (Huang et al. [2016]), social robots (Barakova et al. [2013], Glas et al. [2012, 2016]), industrial manipulators (Stenmark et al. [2017]) and mobile manipulators (Alexandrova et al. [2015], Huang and Cakmak [2017]). The most widely explored approach uses Programming by Demonstration (Sec. 2.3.4). A majority of the work focuses on directly learning a policy or modeling higher level actions from lower level control signals (Akgun et al. [2012], Calinon and Billard [2009], Schaal et al. [2003], Schulman et al. [2013]), while some explore learning task goals or task structure represented in various ways (Ekvall and Kragic [2008], Jansen and Belpaeme [2006], Mohseni-Kabir et al. [2015], Niekum et al. [2013], Pardowitz et al. [2007]). Most closely related to our approach, Akgun and Thomaz [2016] explored simultaneous learning of actions and goals by demonstration, focusing on manipulation tasks, such as closing a box and pouring beans into a bowl.

Robot shelf stacking was part of the Amazon picking challenge in the last iteration (Correll et al. [2016]). Teaching robots to tidy up shelves was addressed previously by Abdo et al. [2015], but their focus is on dividing different products into categories, rather than configuring them on a shelf.

We argue for using end-user programming for teaching task goals and robot actions to perform arrangements of grocery items on shelves. Shelf arrangements are different in every store; only the store owners or staff can correctly specify the task goals for the robot. Hence, the use of end-user programming is essential for that part of the problem. Previous work has explored alternative interfaces, such as speech or different GUIs for specifying task goals (Alexandrova et al. [2015], Kurenkov et al. [2015], Nguyen et al. [2013]). On the other hand, programming of actions is not necessarily tied to end-user programming. An alternative is to give robots universal capabilities for grasping any object, planning a motion to move it without collisions, and placing it in any desired configuration. Motion plans could further involve non-prehensile actions (Dogar and Srinivasa [2010], King et al. [2015]) to reconfigure objects closer together as in some of the actions programmed by demonstration in this chapter. While a lot of research in robotics focuses on giving robots those capabilities, they are still far from being universal. Another approach is for the robot to learn those actions by training on the job *i.e.*, self-exploration using reinforcement learning, but is likely undesirable due to long exploration times and negative impacts of trial errors.

6.2 Approach

Our approach aims at enabling users to demonstrate part of the task of arranging an item on a shelf and have the robot complete the task on its own. From the few user demonstrations, the robot needs to infer both *what* the desired arrangement of objects is and *how* to use its end effector to configure objects in the desired relative configuration. While both of these are well studied problems, their joint inference is common in previous work (*e.g.*, Akgun and Thomaz [2016]). The specific challenge addressed in this chapter is the efficient inference of the task goal from few demonstrations that convey both goal and action. To that end, we explore two practical ideas:

1. We perform a thorough analysis of the task domain to come up with compact domain-specific task representations that exploit common structure.
2. We augment the user’s input with direct specification of certain task parameters that are less efficiently conveyed through demonstrations.

We detail this approach in the following subsections.

6.2.1 Freiburg Dataset Analysis

The Freiburg dataset (Jund et al. [2016]) contains 4,947 images of common grocery items and was originally collected for training visual classifiers. We labeled the images according to product type and shelf arrangement. For the shelf arrangement data we excluded 998 (20.2%) images that either did not show a supermarket shelf (*e.g.*, product on the table or vending machine) or that were duplicate images of a product on a shelf already included. The remaining 3,949 were used for understanding the structure of shelf arrangement tasks and defining a common task representation (Sec. 6.2.2).

Our first observation is that almost all items in stores are arranged on a 2D or 3D *grid*. We considered *rows* to be the depth (front-to-back), *columns* the width (left-to-right), and *stacks* to be the height (bottom-to-top) of the grid. As the dataset focused more on product types rather than product configuration, the configuration parameters were sometimes not all clearly identifiable. The number of rows was often not visible but likely to be filled for the entire depth of the shelf. Thus, we coded the specific number of columns and stacks and assumed rows to be *as many as possible*.

Product categories. Product packaging is often designed with the objective of compact packaging and efficient use of shelf space in stores. As a result a large portion of items (54.3% in our dataset) have a flat top surface that allow *stacking*, of which 33.9% are rectangular prisms (*i.e.*, boxes) and 20.4% are cylinders (*i.e.*, cans). Non-stackable items also have rectangular

or circular bases to stand stably on the shelf surface, but less regular tops (*e.g.*, bottle lids) that prevent stable stacking (Fig. 6.2). We consider these objects equivalent to square pyramids (19.5%) and cones (16.3%). Only 10% of the items did not fit into the shape categories mentioned above and were categorised as soft packaging (8.7%) or other (1.3%), *e.g.*, triangular pyramid or hexagon prism. Table 6.1 shows the full distribution of the dataset across the categories of product types.

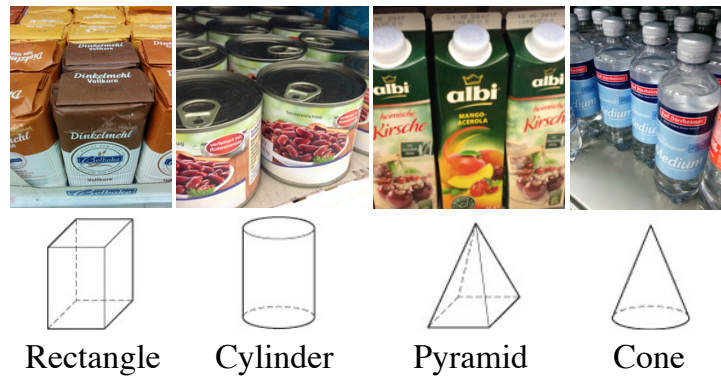


Figure 6.2: Main product shape categories, that comprise 90.1% of items in the dataset, are defined based on the the shape of their base and their tip. The shape of the tip determines whether the objects can be stacked or not. The shape of the base determines whether the object is more compactly arranged on a regular grid (rectangular) or an off-grid arrangement (circular).

Table 6.1: Distribution of product types found in the Freiburg dataset.

Product type category	Count	%
Rectangle (stackable, on-grid)	1676	33.9%
Cylinder (stackable, off-grid)	1011	20.4%
Square pyramid (not stackable, on-grid)	964	19.5%
Cone (not stackable, off-grid)	804	16.3%
Soft packaging: <i>e.g.</i> , candy	428	8.7%
Other: <i>e.g.</i> , triangular pyramid, hexagon prism	64	1.3%
Total	4947	100%

We also observe that items with rectangular bases are most compactly arranged on a grid with both side surfaces touching the neighbouring items. Whereas objects with circular bases can sometimes be more tightly arranged when two consecutive rows are offset by half the radius length of the item, which we refer to as an *off-grid* arrangement (Fig. 6.3a,b). Despite the compactness advantage, only a small percentage (1.75%) of the dataset included off-grid configurations. Although a majority of items were observed to be arranged as compactly as possible (touching neighbouring items on all sides), some arrangements left space between

objects, possibly to allow customers to take items without knocking down others. We observed that at least 0.26% of arrangements left space between items either on the row or column direction (not possible in the stacking direction due to gravity).



Figure 6.3: Cylindrical item arranged in a) regular grid and b) off-grid configurations. c) Soft-packaged item arranged in interleaved grid configuration.

6.2.2 Shelf Arrangement Representation

Based on the common product arrangements observed in the dataset, we propose a simplified grid-based task representation with the following variables: number of columns (m), rows (n), stacks (s), row distance (d^n) and column distance (d^m), where $m, n, s \geq 1$ and $d^m, d^n \in \{0, 1\}$ represent the binary state for touching and not touching respectively. Thus, we represent a shelf arrangement task (where the shelf is orthogonal to the robot) as a tuple $\tau = (n, m, s, d^n, d^m)$. Table 6.2 shows the distribution of the values for these variables observed in the Freiburg dataset. Note that this representation excludes soft-packaged or irregularly shaped items, as well as off-grid arrangements. However, the representation could easily be extended for more complex tasks, such as continuous values for d^n and d^m , variables describing off-grid configurations, or variables for orientation of the items.

Table 6.2: The distribution of shelf arrangement parameter values across the Freiburg dataset.

		stacks					Total
		1	2	3	4	>4	
columns	1	49.4%	4.4%	0.7%	0.2%	2.3%	57%
	2	29.6%	3.4%	0.4%	0%	1.4%	35%
	3	5.8%	0.5%	0%	0%	0%	6%
	4	1.3%	0.2%	0%	0%	0.1%	2%
	5	0.1%	0%	0%	0%	0%	0%
	6	0%	0%	0%	0%	0%	0%
	7	0%	0%	0%	0%	0%	0%
Total		86%	8%	1%	0%	4%	100%

6.2.3 Goal Inference from Demonstration

The problem of goal inference is the estimation of most likely shelf arrangement based on the input obtained from the user. We represent this as a maximum a posteriori estimation problem, *i.e.*,

$$\tau^* = \arg \max_{\tau_i \in T} P(\tau_i | D) \quad (6.1)$$

where T is the set of all possible shelf configurations and $D = \{A_j, O_j\}_{j=1}^M$ is the set of M demonstrations provided by the user with A_j actions, each leading to the placement of one additional object on the shelf. Hence, each O_j corresponds to configurations $(x, y, \theta, s) \in (\mathbb{SE}2 \times \mathbb{N})$ (s for stack number) of j objects that have been placed so far on the shelf surface or on top of another object. Since the demonstrations are assumed to be progressions of the same shelf configuration task, the goal inference only depends on the latest shelf configuration O_M rather than all of D .

Given the M object configurations in O_M we estimate the current shelf arrangement $\tau_M = (n_M, m_M, s_M, d_M^n, d_M^m)$ as follows. The number of rows and columns are determined by separately finding alignments (*i.e.*, values within a small distance) in x and y dimensions of the objects. The number of aligned groups gives the number of estimated rows and columns (n_M, m_M) . The number of stacks (s_M) is estimated as the highest stack number of any object in the demonstrated configuration. The contact between rows and columns (d_M^n, d_M^m) are estimated based on the majority relation between rows and columns.

Next, we substitute τ_M for D and use the Bayes Rule to calculate the posteriors of the term we would like to maximise as follows:

$$P(\tau_i | \tau_M) = \frac{P(\tau_M | \tau_i) P(\tau_i)}{\sum_{\tau_j \in T} P(\tau_M | \tau_j) P(\tau_j)}$$

We assume that the probability $P(\tau_M | \tau_i)$ is zero for all τ_i whose row, column, and stack parameter values are already exceeded in τ_M . Similarly, any arrangement τ_i whose binary variable (d^n, d^m) is contradicted in the demonstration is considered zero. Note that d^n and d^m are only relevant if there are at least two rows or columns respectively ($n > 1, m > 1$). We initialise the priors $P(\tau_i)$ based on their observed occurrence in the dataset (Table 6.2). To have a finite set of possible shelf configurations we bound the grid parameters based on the dataset as well, where $m \in [1..7]$, $n \in [1..2]$ and $s \in [1..5]$, where $n = 2$ corresponds to having *as many rows as possible* based on the shelf, rather than an exact number of rows.

While the computation in Equation 6.1 gives a single most likely configuration, we propose to give the top few arrangements with the highest likelihoods as *suggestions* to the user. This requires ranking all possible arrangements in terms of their likelihoods, given the demonstration. Since ties are possible due to equal priors we rank configurations with smaller parameter values as higher.

6.2.4 Goal Inference with Direct Specification

To allow the user to directly specify goal shelf arrangement parameters to augment demonstrations, we exploit the way that our goal inference works as described in Section 6.2.3. If the user directly specifies the value of a goal arrangement parameter, all arrangements that are inconsistent with that specification are considered to have zero probability. This corresponds to a slight change in the formulation, where the input from the user relevant for the inference does not only include τ_M but also any direct specification \mathcal{S} , where $P(\tau_i|\mathcal{S}, \tau_M)$ is zero if τ_i is inconsistent with \mathcal{S} . This allows for a fast elimination of many candidate arrangements.

6.2.5 Action Representation and Inference

We use a simple action representation proposed in previous work on keyframe-based PbD (Akgun et al. [2012], Alexandrova et al. [2014]). We assume that the robot can perceive the configuration of the shelf and of each object on it. Each of these perceived entities are considered as potential *landmarks*. The action is then represented as a sparse sequence of end-effector poses relative to one of the landmarks and gripper states (open/close). For example, placement of an object next to another object could be done by moving the gripper to a pose near the reference object, lower the gripper, open the gripper, clear the gripper from potential collisions. All of these poses would be relative to the reference object. Non-prehensile actions, such as moving the placed object closer to the reference object, could also be easily represented in the same way, as poses of the gripper relative to detected objects.

Such actions can be directly programmed with a single demonstration using heuristic assignment of poses to landmarks. The assignment can then be corrected by the user (Alexandrova et al. [2014]) or through multiple demonstrations of the same step (Niekum et al. [2012]). We take the first approach; hence an individual action demonstration in A_j is already an executable action. While there can be ways to combine multiple demonstrations, in our implementation the robot randomly chooses an action from the subset of A_j that is reachable based on the configuration of landmarks in the scene. Although all demonstrated actions should be reachable for the objects in the shelf arrangement that were placed during the demonstration, some actions may become unreachable when they are shifted for placement of other objects to complete the desired shelf arrangement. Hence having multiple demonstrations in A_j is still valuable.

Execution of actions simply involved detecting all landmarks in the environment, computing the actual configuration of end-effector poses that are relative to those landmarks, and then moving through the landmarks one by one.

6.2.6 Implementation

We implemented our approach on a Fetch mobile manipulator which has a 7-DoF arm with a load capacity of 6kg. Our system builds on the open-source system Rapid PbD (Huang [2018a]) which implements the action representation described in Section 6.2.5. The robot detects the shelf surface and objects on it using standard Point Cloud Library functionality. As stacked objects are perceived as one cluster on the surface, we detect stacking from height changes that are approximately equivalent to the height of the object measured at the demonstration of the first action.

We extend the Rapid PbD graphical user interface (GUI) in several ways. Users first create an empty program using the GUI. Then they kinesthetically move the robot arm to desired poses and save poses or change the gripper state through the GUI. We assume that the items to be placed on the shelf are always picked up from the same location. Demonstrations involve a sequence of picking, moving, placing, and possibly pushing for non-prehensile readjusting of one object relative to the shelf or to another object already on the shelf.

When the user confirms the end of an action demonstration the robot perceives the end state of the demonstration O_M , which is expected to include M objects. The robot then infers τ_M as described in Section 6.2.3 and visualises the most likely configuration overlaid onto the detected objects that are already placed (Fig. 6.4). In our implementation the average row and column distances ($d^m, d^n \geq 0$) are read from the demonstrated arrangement and off-grid arrangements are recognised. The user can create interleaved arrangements (Fig. 6.3c) by changing d^m, d^n on the GUI. It also displays the top four arrangements that have the highest likelihood given the demonstrations so far. The GUI allows the user to select an arrangement from this list or directly specify task goal parameters (Sec. 6.2.4). When the goal for the shelf arrangement is confirmed, the robot can execute the complete shelf arrangement task either continuing from what has already been demonstrated or starting from scratch.

Our interface also allows users to reuse previously programmed tasks and actions for shelf arrangement with different objects or grid parameters. To that end, the user can set up full or partial shelf arrangements and then run the object detection and goal inference. They can also change task parameters to match the new task. Programmed manipulation actions can also be transferred in some cases, but need to be tested and possibly adjusted to work with different objects.

6.3 Goal Inference Evaluation

We evaluate our approach in three ways. First, in this section we present (1) a systematic evaluation of the goal inference on the Freiburg dataset and (2) a user evaluation through an online study with a simplified interface. Next, in Section 6.4 we present the full system evaluation.

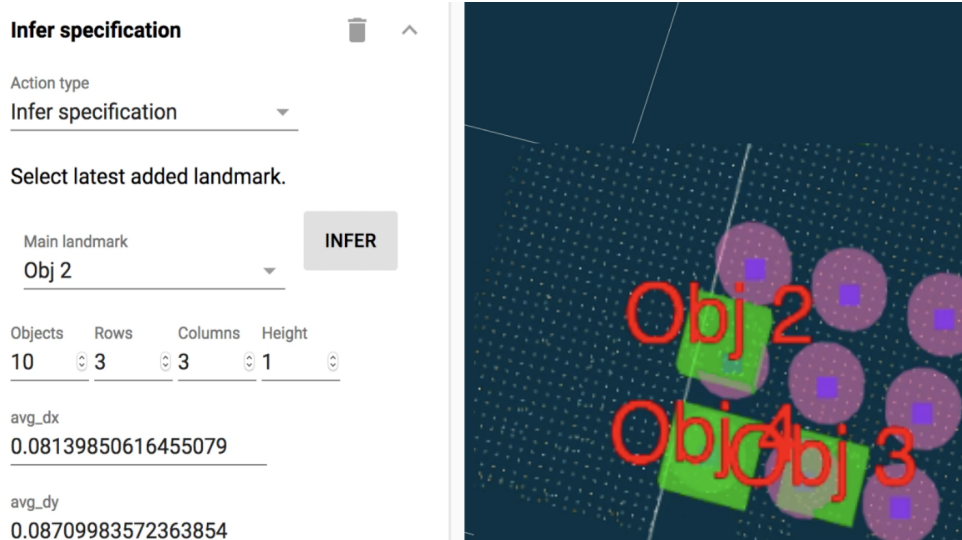


Figure 6.4: The extended Rapid PbD interface where users can directly specify their desired object arrangement (left) and the inferred arrangement is visualised (purple) overlaying the detected objects (green).

6.3.1 Teaching Strategies

Our system provides the user with different possibilities for programming shelf arrangement tasks. As described in Section 6.2.3, at any point during the programming process the user has three possible actions:

- *Demonstrate* object placement kinesthetically,
- *Specify* grid parameters on the interface,
- *Select* an arrangement from most likely arrangements proposed on the interface,

The user can have different strategies for using a combination of the different actions to minimise costs. Often times, learning algorithms are evaluated with *sample efficiency*, *i.e.*, the number of examples needed to learn a concept. Similarly, users of our system can try to minimise the number of programming actions they need to take in order to correctly teach a shelf arrangement. Alternatively, each action can be associated with a different cost, such as *clock time* and the user can try to minimise it. Given the three programming actions above, we devised the following programming strategies:

1. Naive demonstration (ND) involves demonstrating the full shelf arrangement filling up rows, columns, and stacks in progression.
2. Optimal demonstration (OD) involves giving the minimum number of demonstrations to make the inference correct; *i.e.*, filling up one row, one column and one stack.

3. Demonstrate and specify (D&S) involves demonstrating distances d^m and d^n and specifying remaining variables n, m, s .
4. Naive specification (NS) involves specifying all values in order of n, m, s, d^n, d^m .
5. Prior-aware specification (PaS) involves only specifying values that are different from the most likely values according to the prior.

These programming strategies are akin to optimal teaching algorithms that can be devised for a given concept and known learner (Cakmak and Lopes [2012], Khan et al. [2011]). The *Select* action can be used in combination with any of these strategies, therefore resulting in 10 teaching strategies. We assume that the interface displays the top four most likely configurations. Hence, if the most desired arrangement is in the top four, the user can directly select this configuration and complete the programming process. Note that in order to learn the action as well as the task goal, the robot needs to obtain at least one demonstration from the user, which is not the case for all strategies. However, for the purposes of the analysis of goal inference, we ignore that fact. In practice, those strategies that do not involve any demonstrations would require at least one demonstration in addition to the other programming steps.

6.3.2 Evaluation on the Freiburg Dataset

We measured the performance of each teaching strategy in terms of the number of steps (*i.e.*, actions) required to infer the desired arrangement across the Freiburg dataset. Figure 6.5 shows the learning curves for the ten teaching strategies for four example shelf configurations, where the ground truth is represented as described in Section 6.2.2. We observe that for simpler configurations ($m, n, s \leq 2$), strategies that involve demonstrations are comparable or better than specification strategies, because (1) d^n, d^m are automatically inferred from the object placements and (2) our priors are based on the dataset where simpler configurations are ranked higher (Sec. 6.2.3).

For higher m, n, s values, the naïve demonstrations become infeasible, optimal demonstrations are bounded by $(n + m + s - 2)$ steps, while specification strategies are always bounded by a maximum of 5 steps. The version of each strategy that involves a final selection step from the top four arrangements is more efficient or the same (in terms of number of actions) as the original strategy, since it takes fewer demonstrations or specifications to get the desired arrangement in top four versus top one. The most efficient strategy is the *prior-aware specify+select* strategy requiring 1.85 steps on average (Fig. 6.6). Overall, the average number of steps required to teach shelf configurations for all teaching strategies is low (between 2-3 actions) despite some outlier cases (*e.g.*, *naive demonstrations* requiring over 50 demonstrations (Fig. 6.5d)). This is a positive outcome of using priors that are based on the dataset, which in turn allows faster inference on the majority of the data.

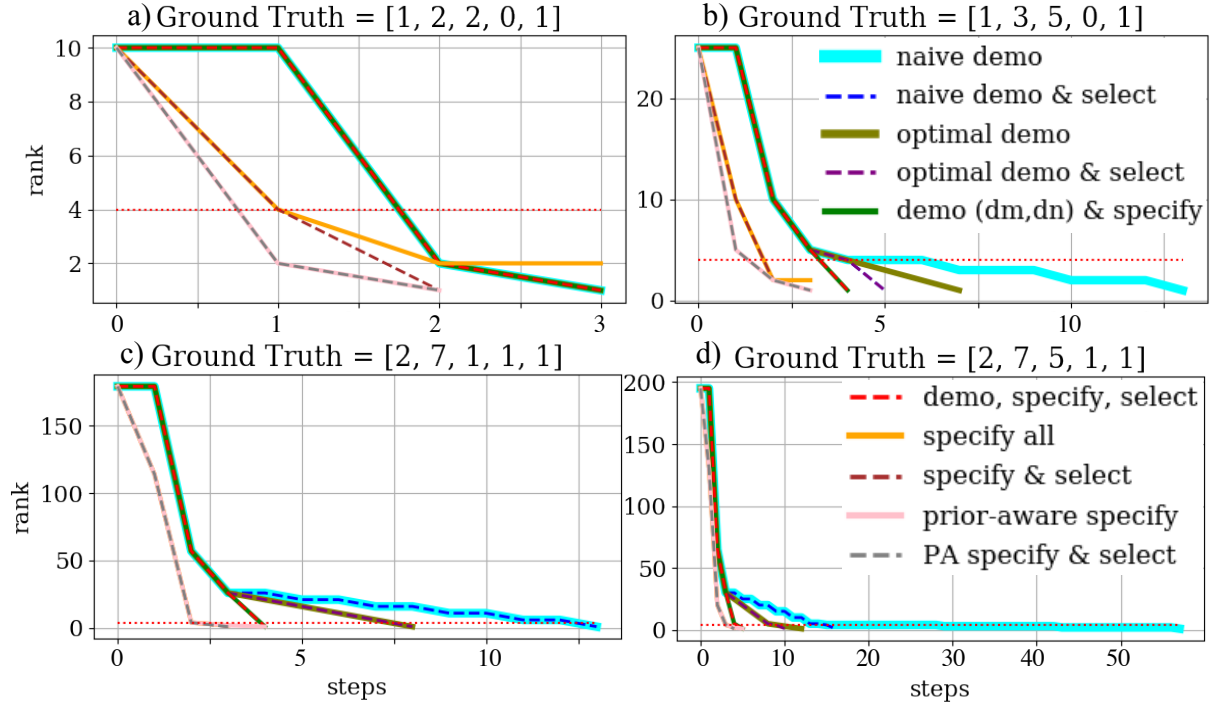


Figure 6.5: Learning curve of 10 teaching strategies for 4 different shelf arrangements from the Freiburg dataset, with number of actions (x-axis) and ranking of the ground truth shelf arrangement (y-axis).

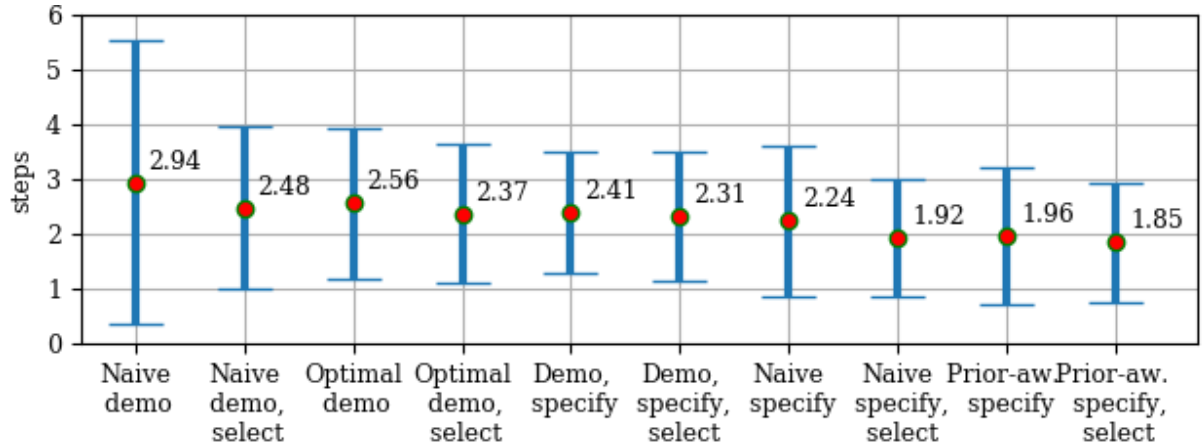


Figure 6.6: Average number of steps required to infer the correct arrangement per teaching strategy across the dataset (error bars show standard deviation).

6.3.3 User Study Evaluation

Next we conducted a user study on Amazon Mechanical Turk (AMT)¹, an online crowd-sourcing platform used to solicit and compensate participants for user studies (Kittur et al. [2008]). The aim was to evaluate the goal inference method and investigate teaching strategies preferred by human users. For this, we created a graphical interface (Fig. 6.7) for a simplified 2D arrangement task domain, where users had to instruct the robot how to arrange grocery store items by row and column. We were interested in measuring how quickly the desired goal configuration could be inferred from user inputs, what programming actions were preferred, and how they performed in comparison to the teaching strategies defined in Section 6.3.1. The experimental material such as source code for the interface can be found in Appendix D.

6.3.3.1 Protocol

First users were shown a brief instruction video on how to perform the different programming actions on the interface. Then the user moved onto programming desired arrangements, which were communicated to them through a grocery store picture. The user interface, shown in Figure 6.7 had three parts for performing the three programming actions. Demonstrations were performed by dragging and dropping items onto a shelf area, which automatically updated the specification fields with the inferred values of task parameters. Specifications were done through drop down menus. The visualisation of the most likely top four arrangements was updated after every user action. The user could select one of these by clicking on it. Participants were told to confirm the inferred arrangement once it matched the desired arrangement. Each user completed one practice task and 8 arrangement tasks in a randomised order. After the final task, participants were asked to fill out a questionnaire to provide further insight into their preferred actions and strategies.

6.3.3.2 Metrics

We recorded the user’s interactions with the interface, when they performed a demonstration, when they changed the value of a specification field, when they selected one of the proposed arrangements, and when they confirmed the inferred arrangement. We used these recordings to measure the user’s preferred action and strategy, the number of steps and time spent per task, and if the inferred arrangement was correct. In the survey at the end, we asked users about their most and least preferred teacher actions, to explain their preference and strategies, and to rate the usefulness of the proposed likely arrangements on a 4-point Likert scale.

¹www.requester.mturk.com

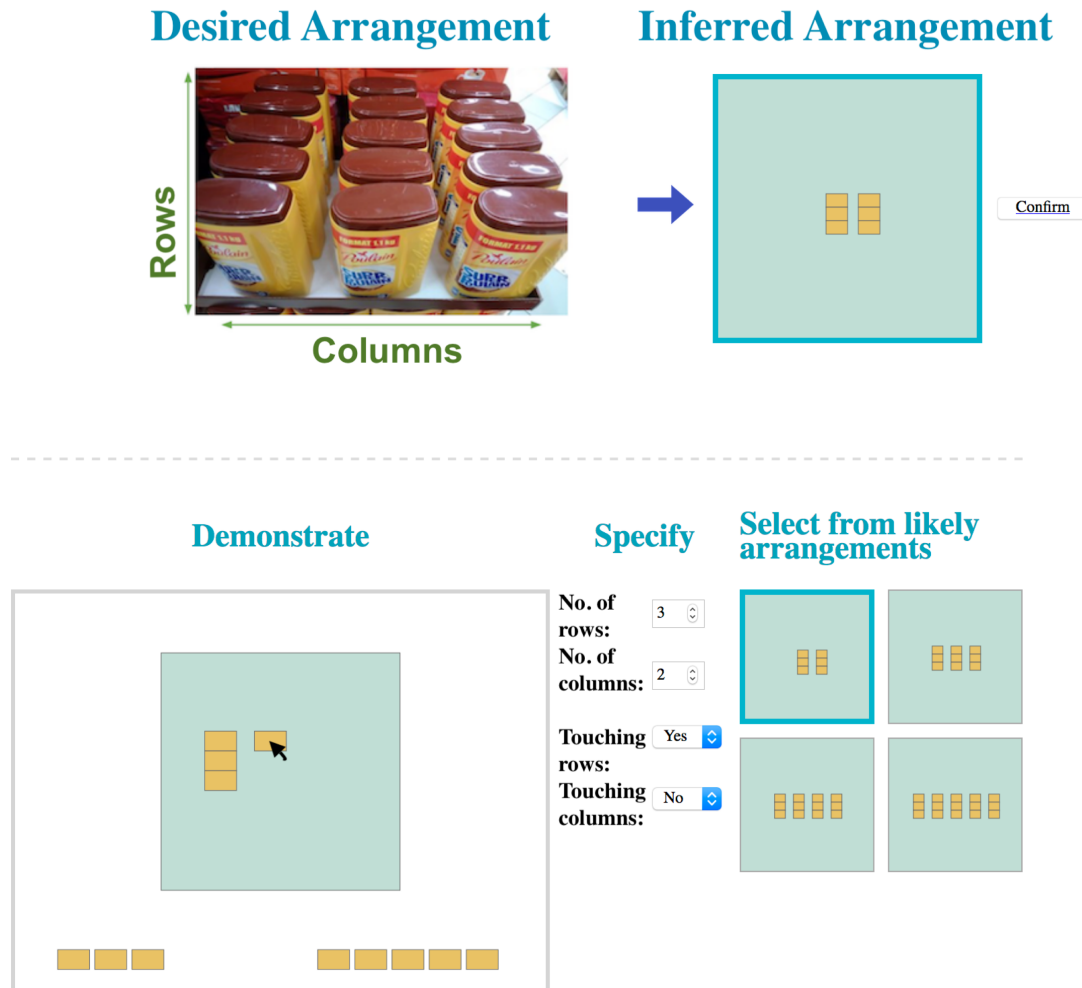


Figure 6.7: Simplified user-interface created to evaluate our goal inference model in an online user study. The top part shows the desired configuration with a picture and the visualisation of the current shelf arrangement inference. The bottom part has three parts corresponding to three types of programming actions the user can take: demonstration (drag-and-drop items onto shelf), specification (select parameter values from drop-down menus), and selection (choose one of the top four most likely arrangements by clicking on it).

6.3.3.3 Results

The study included 32 participants (21 male, 11 female) with an average age of 32.5 years (SD=9.6). Users took an average of 26.5 seconds (SD=7.8) to complete a task.

User performance. Participants correctly completed an average of 5.6 tasks (SD=1.8), 1.8 tasks (SD=1.6) with correct number of rows and columns but wrong distances, and 0.7 (SD=0.9) incorrectly. The wrong distances were likely caused by the angle of the example pictures, as some users did not consider objects as touching, *e.g.*, when they were cone-shaped. Other mistakes were likely due to mis-counting rows or columns. Figure 6.8 shows the average number of steps taken by participants to complete a task in comparison to the teaching strategies defined in Section 6.3.1. Overall users were slightly more efficient than the optimal demonstration (OD) strategy, but worse than most other strategies.

User strategies. Figure 6.9 shows the strategies employed by participants across the different shelf configurations they programmed. We see that using naive specifications (NS) was most common, followed by two strategies that combined specifications with the other actions. Most users (71.9%) stated their most preferred action to be *specify* and the least preferred to be *demonstrate*, with 16 (50%) out of 32 users explaining that they found specification to be “easier”. The majority (84.4%) of the users found the top 4 proposed arrangements to be useful, but the *Select* action was not used as much as *specify*. Most users (28.1%) who described their teaching strategy stated *specify & select*; *e.g.*, “type in the rows and columns and then pick the picture that was best”. Few users (9.4%) stated their strategy to be a combination of the three actions as shown in the tutorial video. One user mentioned a prior-aware strategy: “I liked to specify exactly what I wanted, but I also could have adjusted the proposed configurations a little bit”.

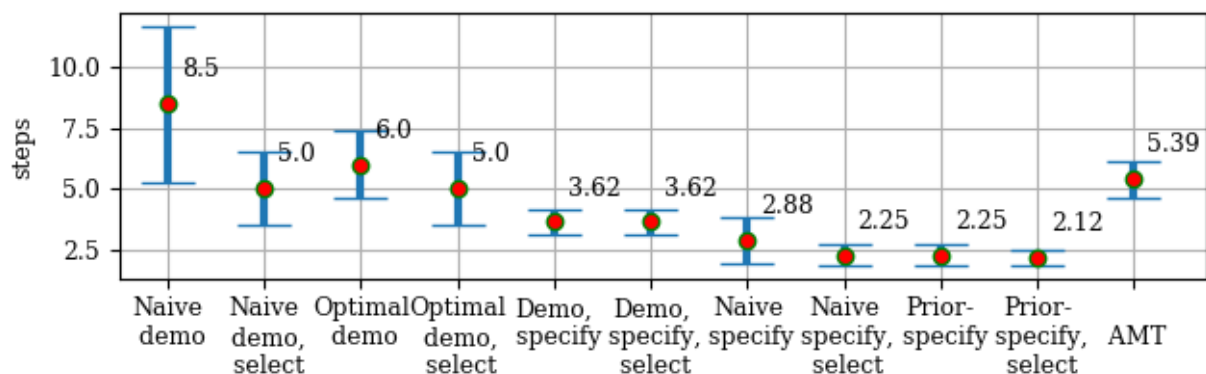


Figure 6.8: Average number of steps required for 8 arrangement tasks chosen for the online user study, comparing 10 teaching strategies with human performance (AMT).

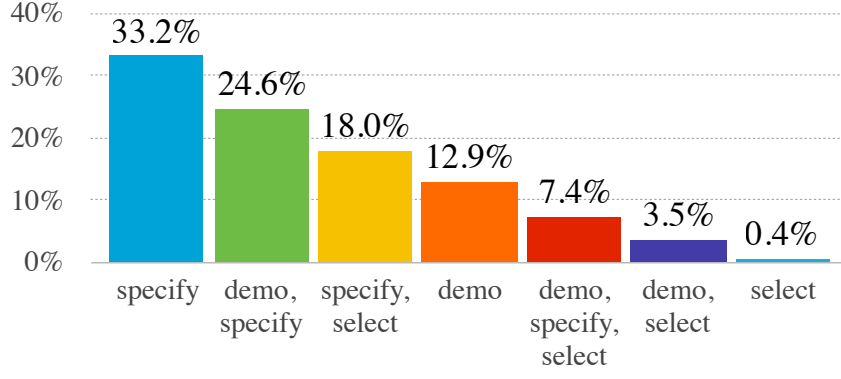


Figure 6.9: Distribution of user strategies employed in the AMT study

6.4 System Evaluation

To demonstrate our full system’s ability to learn and execute shelf arrangement tasks, we defined a list of benchmark tasks described in Table 6.3. The tasks varied in product type and grid configuration parameters, some of which required different manipulation actions (*e.g.*, to place objects in touching configurations while avoiding collisions of the gripper with other objects). We also included more complex arrangements such as off-grid cylinders and interleaved soft packaging (Fig. 6.3b,c) which were excluded from our goal inference analysis due to low frequency in the dataset.

Table 6.3: Benchmark tasks used for evaluating the full system implementation on the Fetch robot.

#	Grid (n,m,s)	Used manipulation actions	Product
1	(4,1,1) not-touching	Pick and place	Tin cans
2	(2,2,2) not-touching	Pick and place	Tin cans
3	(1,4,1) touching	Pick, place, push left	Tin cans
4	(1,3,1) close distance	Pick, place, push left	Cereal boxes
5	(1,3,1) close distance	Pick, place, push left	Toothpaste
6	(3,3,1) off-grid	Pick, place, push left&front	Tin cans
7	(3,3,1) off-grid	Pick, place, push left&front	Soda cans
8	(2,3,1) interleaving	Pick and place from top	Candy bags

6.4.1 Protocol

All tasks were programmed as efficiently as possible using the teaching strategies from Section 6.3.1 and reusing previously taught actions. For each task, the experimenter decided what type of manipulation action was required depending on the object type and the grid configuration. The experimenter could also decide if a new action needed to be demonstrated or if a previously existing one could be reused (as described in Sec. 6.2.6). The actions were chosen to be as simple as possible for the given task, omitting any unnecessary gestures or movements (*e.g.*, if a simple place action was sufficient, no additional push action should be taught). When programming an action, the experimenter could let the robot execute the partial arrangement task for a subset of objects as part of the programming process to mitigate errors in the final task execution.

6.4.2 Results

We programmed four different manipulation actions to complete the eight benchmark tasks. Manipulation actions included pick, place (from front), place from top, push left, and push left&front. We were able to reuse these actions for similar tasks by modifying the task configuration and program steps. Each task was executed successfully from start to end at least two times. Figure 6.10 shows snapshots from the action executions of the benchmark tasks. The evaluation demonstrates our system’s capability to learn manipulation actions for the main product types, as well as soft packaging (*e.g.*, candy), which cover 98.7% of the Freiburg dataset².

6.5 Discussions

In the AMT study, users performed demonstrations using drag-and-drop operations which is simpler than stacking objects on a shelf with a real robot. As users preferred specification strategies over drag-and-drop demonstrations, it is likely that demonstrating object stacking would be less preferred as well. Some limitations of our approach are as follows.

1. Our system only considers one demonstrated manipulation action, even if multiple demonstrations are performed. An extension could consider the poses of multiple action demonstrations and infer an action adapted to a specific scenario (*e.g.*, only use push action if the gripper would collide with other landmarks).
2. The system’s perception system is limited as it does not recognise separate objects that are too close together. This limits our ability to detect demonstrated configurations for objects that are meant to be touching.

²Sample executions of shelf arrangement tasks can be seen at <https://youtu.be/liaSirH0CeM>

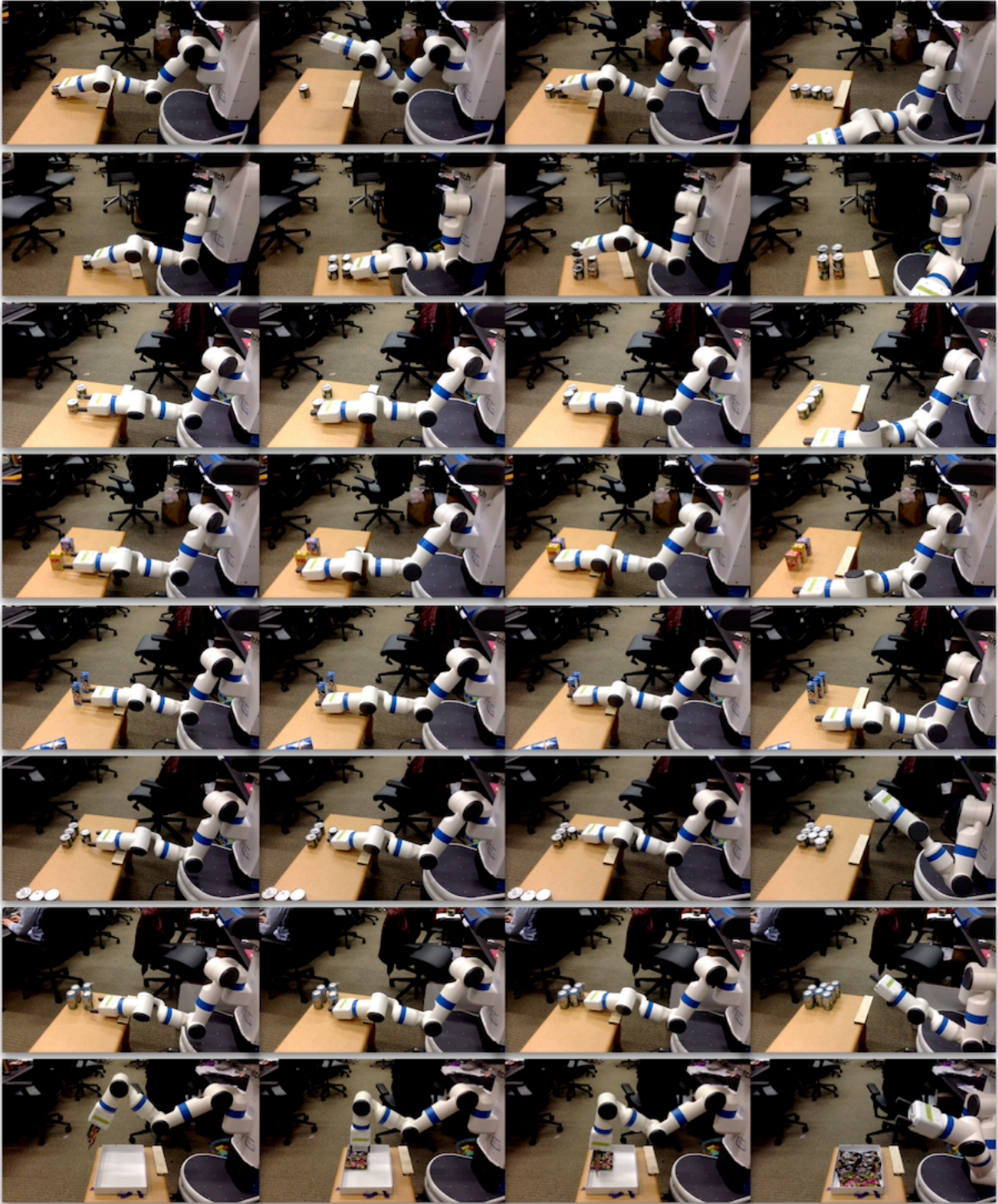


Figure 6.10: Snapshots from the executions of the eight system evaluation benchmark tasks.

3. We did not consider the orientation of the object (*e.g.*, product label facing front) as commonly applied in shelf organisation tasks. Our perception system only recognises the object’s bounding box, so an extension could include better object recognition of object orientations.

6.6 Conclusions

In this chapter we presented a goal-oriented end-user robot programming system to efficiently teach a robot shelf arrangement tasks and actions. We proposed augmenting Programming by Demonstration with domain-aware goal inference and direct user inputs to accelerate the teaching process. Our task goal representation is based on an analysis of a large database of grocery store shelf images. We evaluated our goal inference with different teaching strategies and with real human teachers in an online user study on AMT. We then demonstrated our system’s ability to efficiently learn and perform various shelf arrangement tasks and actions on a Fetch mobile manipulator.

In the context of this thesis, we demonstrated the generalisability of our chosen goal-oriented robot programming approach using keyframe-based PbD and validated its usability for end-user applications. We demonstrated the system’s expressivity by allowing users to easily teach robots a variety of customised actions with the help of a graphical user interface. The implemented system is the first step towards creating the robot programming framework proposed in Chapter 4. It includes both teaching the robot low-level actions by demonstration and a graphical interface to facilitate the end-user programming process. The main focus for the remaining thesis is a complete implementation of the proposed framework and augmenting the graphical interface to allow intuitive navigation of the different functionalities. Building on top of this system, the remaining steps to implement are learning high-level action representations and integrating the task planner resulting in an end-to-end goal-oriented robot programming system.

Chapter 7

End-User Robot Programming for Task Planning

Contents

7.1	Related Work	95
7.2	Approach	96
7.2.1	Low-level Action Representation	96
7.2.2	High-level Action Representation	97
7.2.3	Action Inference from Demonstration	97
7.2.4	Action Generalisation	99
7.3	System Implementation	99
7.3.1	Interactive Robot Programming	99
7.3.2	Plan Execution	101
7.4	System Evaluation	101
7.4.1	Protocol	102
7.4.2	Results	102
7.5	User Evaluation	103
7.5.1	Experimental Setup & Participants	103
7.5.2	Experimental Design & Measurements	104
7.5.3	Results	105
7.5.4	Continuous Improvement of the System	108
7.6	Discussions	110
7.7	Conclusion	110

In the previous chapters we completed preliminary steps to validate that end-users can easily learn the main concepts of the automated planning language and that the proposed robot programming process is intuitive (Chapter 5). We then implemented a goal-oriented robot programming system that uses keyframe-based PbD and validated its usability to teach a variety of low-level actions (Chapter 6). Our main focus now lies in the full implementation of the iRoPro framework proposed in Chapter 4. In this chapter we present an end-to-end system implementation for teaching robots primitive actions to be reused with a task planner (Fig. 7.1). The robot simultaneously learns low- and high-level action representations from demonstration and infers a generalised action that can be reused directly with a task planner to solve problems that go beyond the initial demonstration.

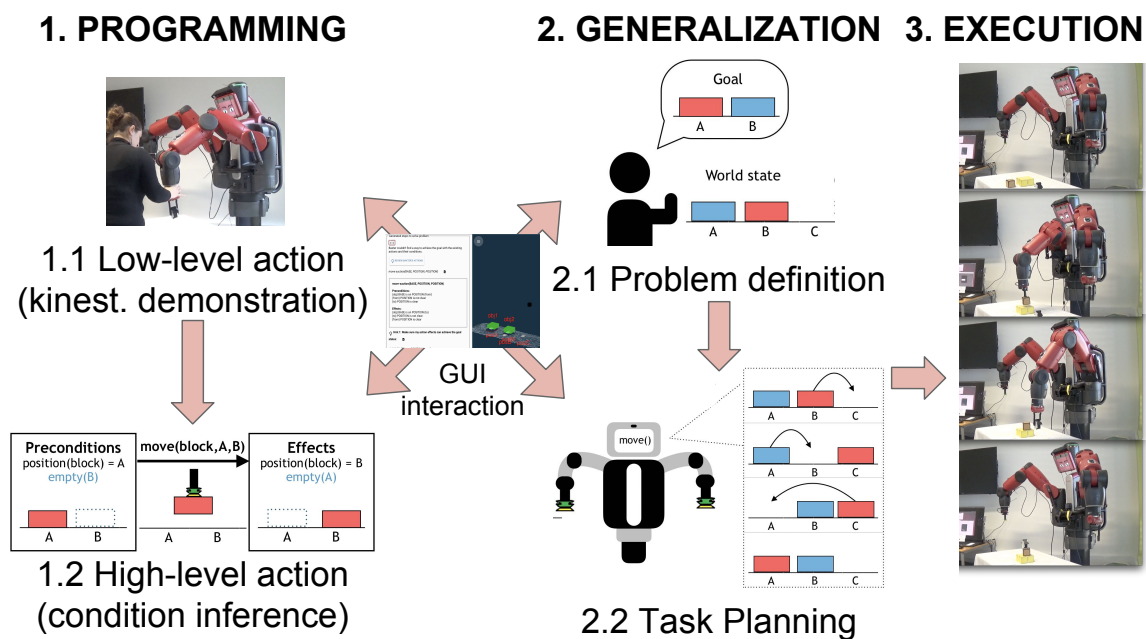


Figure 7.1: Overview of iRoPro that allows users to teach low- and high-level actions by demonstration. The user interacts with the GUI to run the demonstration, modify inferred action conditions, create new planning problems for the robot to solve and execute.

In the following we first present some related work, followed by our approach on learning low- and high-level action representations from demonstration to reuse them with a task planner (Sec. 7.2). We then provide details of the system implemented on a Baxter robot (Sec. 7.3). The system includes a graphical interface that allows users to teach new actions by kinesthetic demonstration, modify infer action conditions, define new planning problems, have the robot autonomously solve them and execute the plan in real-time. In Section 7.4 we demonstrate our system’s capability to generalise primitive actions on six benchmark tasks that are programmed and executed on the Baxter robot. In Section 7.5 we empirically investigate the usability of our system and validate its intuitiveness through a study with 21 users of different educational

backgrounds and programming expertise. To better understand user teaching strategies, we split participants into two control groups, with and without automatic condition inference, and investigate how users in both groups learn and use the system. Finally, we discuss limitations and possible extensions to further increase the system’s generalisability (Sec. 7.6).

7.1 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 (Paxton et al. [2017], Perzylo et al. [2016], Stenmark et al. [2017]). For example, Paxton et al. [2017] use Behaviour Trees to represent 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 by Plch et al. [2012]), or common knowledge in software engineering (GIPO by Simpson et al. [2007], itSIMPLE by Vaquero et al. [2013]). There has been previous work on integrating task planning with robotic systems (Cashmore and Fox [2015], Kuhner et al. [2018]), learning high-level actions through natural language instructions (She et al. [2014]) or learning preconditions and effects of actions to be used in planning (Jetchev et al. [2013], Konidaris et al. [2018], Ugur and Piater [2015]). 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.

Programming by Demonstration (Sec. 2.3.4) has been commonly applied to allow end-users to teach robots new actions by demonstration. Alexandrova et al. [2014] 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. [2013] 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 thesis 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.

7.2 Approach

iRoPro 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. 7.2.1) and *when* (Sec. 7.2.2) an action should be applied. To accelerate the programming process, action conditions are directly inferred from a single demonstration (Sec. 7.2.3). The action generalisation is performed on both low- and high-level representations (Sec. 7.2.4), allowing it to be reused with a task planner (Chapter 3). We will describe our approach in the following sections.

7.2.1 Low-level Action Representation

As we build on top of the system presented in Chapter 6, low-level actions are represented similarly using keyframe-based PbD as a sparse sequence of end-effector poses and gripper states (as described in Sec. 6.2.5). 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 Section 7.2.2.

New actions are initialised with the robot’s end-effector poses in a neutral position (as seen in Fig. 7.2) to allow unobstructed object detection. 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.

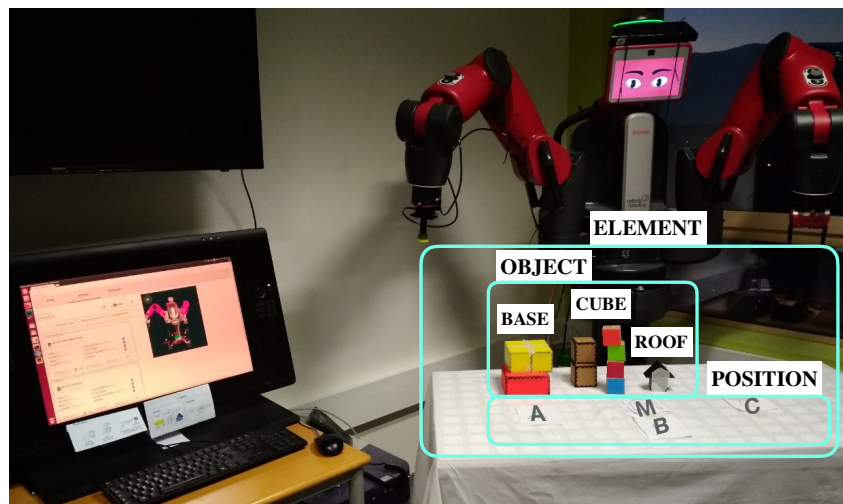


Figure 7.2: Experimental setup for the user study (N=21), where users programmed the Baxter robot via a GUI to manipulate given object types (with predefined type hierarchy).

7.2.2 High-level Action Representation

We represent high-level actions as proposed in task planning (Chapter 3), 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*. 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. 7.2).

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'). We implemented predicates that are commonly used in task planning domains as well as two additional ones to further 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

Note that the set of types and predicates could easily be extended for more complex tasks and that it is possible to detect them automatically. However, this is beyond the scope of our work, so in our implementation CUBE and BASE objects are by default flat, and CUBE and ROOF objects are thin enough for the robot to grasp.

7.2.3 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. Object positions are determined by the proximity of the object to given positions. For example, 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 the action conditions, the robot perceives the world state before and after the kinesthetic action demonstration as seen in similar work for learning object manipulation tasks (Ahmadzadeh et al. [2015], She et al. [2014]). Let $O_b = \{\phi_1, \phi_2, \dots\}$ be the set of predicates observed *before* the action demonstration and $O_a = \{\psi_1, \psi_2, \dots\}$ *after*. The action inference is the heuristic deduction of predicates that have changed between O_b and O_a , *i.e.*,

$$\begin{aligned} \text{pre}(a) &= (O_b - O_b \cap O_a) = \{\phi_i | \phi_i \in O_b \wedge \phi_i \notin O_a\}, \\ \text{eff}(a) &= (O_a - O_b \cap O_a) = \{\psi_i | \psi_i \notin O_b \wedge \psi_i \in O_a\}, \end{aligned}$$

where $\text{eff}(a)$ includes positive and negative effects (Fig. 7.3).

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.*,

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

Note that conditions could be learned from multiple demonstrations (Abdo et al. [2013], Konidaris et al. [2018]). 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.

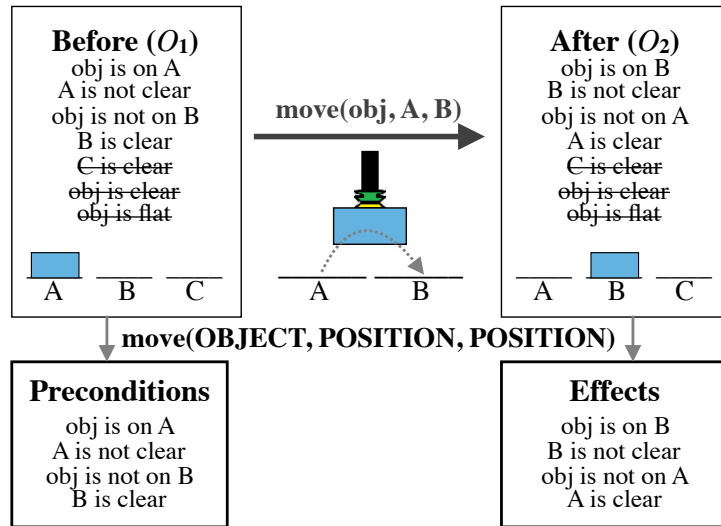


Figure 7.3: Example of a high-level action for moving an object from A to B. Conditions are inferred from the observed predicates before (O_b) and after (O_a) the demonstration.

7.2.4 Action Generalisation

Due to simultaneously learning low- and high-level action representations, an action is generalised in two ways:

1. Low-level: the demonstrated action consists of poses relative to detected landmarks during the action demonstration. To generalise this action to different landmarks, objects are redetected and relative poses are re-calculated.
2. High-level: the action parameter types and preconditions specify what landmarks an action can be applied to. Changing these properties via the GUI allows an action to be generalised and the taught manipulation action to be transferred to other tasks.

7.3 System Implementation

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 each (Sec. 5.1). For the object perception we mounted a Kinect Xbox 360 depth camera on the robot. We developed a user interface as a web application, based on HTML and JavaScript, that can be accessed via a browser on a PC, tablet or smartphone. The user interface provides an abstraction from the underlying planning language and displays predicates in natural language in English.

The source code for iRoPro is developed in ROS (Quigley et al.) and available online (Appendix E). The low-level action is learned using the open-source system Rapid PbD (Huang [2018a]). The integration of the task planner is implemented using the ROS package PDDL planner (Ueda [2018]). 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 (Huang [2018b]). An object is represented by its detected location and bounding box, *i.e.*,

$$obj = (x, y, z, width, length, height)$$

where the bounding box is used to infer the object type. We implemented a partial PDDL domain in iRoPro that includes a set of predefined object types and predicates (Sec. 7.2.2). The user completes the domain by creating new actions via the interface and uses the integrated task planner to solve new problems.

7.3.1 Interactive Robot Programming

The user interacts with the GUI (Fig. 7.4) to visualise the robot and the detected objects, create new actions, run the kinesthetic teaching by demonstration, correct inferred action parameters or conditions, create and solve new problems with the task planner. The interactive robot programming cycle consists of creating and modifying actions and problems.

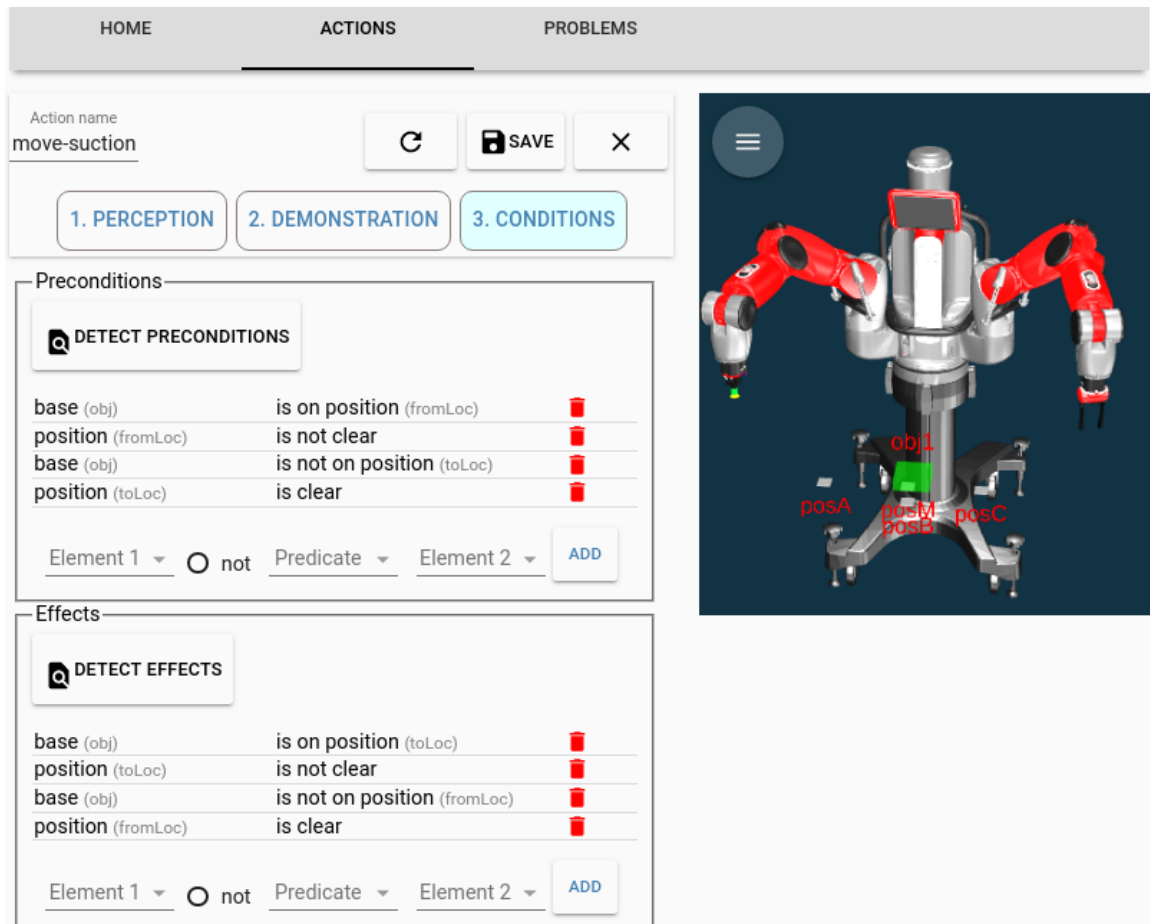


Figure 7.4: The iRoPro interface showing the action condition menu and an interactive visualisation of the Baxter robot and detected objects.

Actions

New actions are taught by kinesthetically moving the robot's arms (low-level action) and assigning action conditions (high-level action). The low-level action is learned by keyframe-based demonstration. To verify the taught action, the user can have the robot re-execute it immediately. The high-level action is inferred by capturing the world state before and after the action demonstration (as described in Sec. 7.2.3). Action generalisation can be done by modifying action properties. 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 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 via the interface if the inference was not correct. Then, the user enters predicates that describe the goal to achieve. The complete planning domain and problem are translated

into PDDL and sent to the Fast-Forward planner (Hoffmann and Nebel [2001]). 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. 7.5) 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 planning problem or modify existing ones by redetecting the objects.

7.3.2 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. 7.2.1). To accelerate the execution, we only detect the landmarks once at the start and save their new positions in a *mental model*, *i.e.*, a temporary memory. 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. 7.6).

7.4 System Evaluation

We evaluate our system’s generalisability on six benchmark tasks (Table 7.1) and show how taught actions can be reused for complex tasks. The tasks involved manipulating different object types on four marked positions with both claw and suction grippers. We take the Blocksworld domain (Slaney and Thiébaux [2001]) 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, *i.e.*, using different grippers. The order of the tasks was given with increasing complexity, requiring the user to modify existing actions or teach new actions from scratch.

Table 7.1: Benchmark tasks for the system evaluation. Three different pick-and-place actions using (claw and suction grippers) 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 & claw from side
5	Build house with BASE, CUBE, ROOF	claw from top & suction from top
6	Rebuild Task 5 on a different position	claw from top & suction from top

7.4.1 Protocol

The tasks were programmed with the most efficient teaching strategy of minimising the number of actions created and generalising them by changing the action properties (as described in Section 7.2.4). 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 states. 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 in real-time. 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.

7.4.2 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’. The robot was able to generate plans for all tasks and executed them in real time at least twice (Fig. 7.5).¹ Note that the taught

¹A subset of the task executions can be seen on <https://youtu.be/NgaTPG8dZwg>

actions can be reused for a diverse range of problems beyond the six benchmark tasks. This evaluation shows the generalisability of our system, allowing us to teach primitive actions by demonstration and reuse them with a task planner to solve more complex problems.

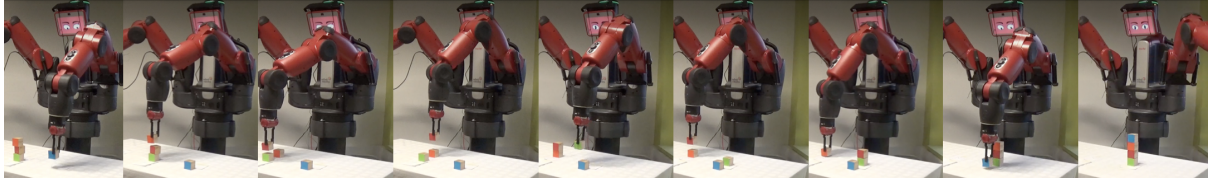


Figure 7.5: Snapshots from the executions of the system evaluation (Tasks 3&4) showing a claw grip from the top and the side.

7.5 User Evaluation

The second and main part of the evaluation was conducted using the THEDRE (Traceable Human Experiment Design Research) method (Mandran [2018], Mandran and Dupuy-Chessa [2017]) which aims to evaluate computer systems in a research context by integrating a user-centered approach. It is based on continuous improvement and takes a pragmatic constructivist approach (Avenier and Thomas [2015]), allowing us to further develop the system as well as the scientific knowledge from the experimental ground. To that end, it offered us the possibility to mix qualitative and quantitative approaches in order to gather as much data as possible to evaluate and improve our system.

7.5.1 Experimental Setup & Participants

User experiments aimed to evaluate our approach implemented on a Baxter robot with real end-users. However, we were also interested in the user's programming strategy of using the system. Thus, we split participants into two control groups, with and without condition inference (Sec. 7.2.3) and observe user strategies for completing the 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 evaluation - Group 1 vs 2: users without automatic condition inference will understand the system better

H5 Pre-test questionnaire: users that perform better in the pre-test questionnaire can learn to use the system faster

The experiments were conducted with a Baxter robot with two grippers (suction and claw) and mounted with a Kinect Xbox 360 camera. A table with 4 marked positions was placed in front of the robot with different objects to be manipulated by the robot during the experiment (Fig. 7.2). Participants had access to a computer with a mouse and a keyboard to program the robot via the graphical interface.

The study was conducted with 21 participants (10 male, 11 female) 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.

7.5.2 Experimental Design & Measurements

Users were first given a brief introduction to task planning concepts, the Baxter robot and the experimental set up. They were then asked to complete a pre-study questionnaire to capture the participant's profile and their 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 7.2). The tasks were designed to familiarise them with different aspects of the system: create new actions (Task 1,6), 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. The experimental protocol, pre- and post-study questionnaires and additional material used can be found in Appendix E.

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, GUI activity log, pre- and post-study questionnaires.

Table 7.2: Benchmark tasks for the user study where the first two tasks were used to introduce participants to the system.

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

The pre-study questionnaire 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 pre-conditions for the given action'). The questions were multiple choice and each question was normalised to count at most 1 point if answered correctly. The highest achievable score was 7.

In the post-study survey we used the System Usability Scale (SUS) (Brooke [1996]) 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 (Tullis and Stetson [2004]). As a benchmark, we included questions from our previous user study in Section 5.3, where users were simulated a robot programming experience using the Wizard-of-Oz technique. Finally, participants were asked which aspects they found most useful, most difficult, and which they liked the best and the least.

7.5.3 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 presented results.

User performance (H1-H3)

Users took an average of 41.2 minutes (STD=9.08) to complete the main tasks (3-8). 'non-CS' users completed the tasks the fastest (AVG=36.6, STD=7.46), followed by users with no programming experience (AVG=43.6, STD=5.37). 'CS' users took on average longer (AVG=43.8,

STD=14.13) as they were often interested in testing the system's functionalities with different inputs.

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 instead of generalising the existing one by changing 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). The generated PDDL code for the planning domain can be found in Appendix F.

Condition inference (H4)

To evaluate user programming strategies, the condition inference (CI) (Sec. 7.2.3) in our study only generated a minimal set of predicates, which did not cover predicates needed for later tasks. We noticed a discrepancy in the programming strategies between the two control groups (Group 1 with CI vs Group 2 without CI). As participants in Group 2 had to add action conditions manually, they considered all predicates they deemed necessary for the action and therefore added additional ones that were required for later tasks. On the other hand, we observed that Group 1 had the tendency to leave the inferred conditions unmodified without adding conditions that were missing. 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 (Group 1: AVG=41, STD=7.89 and Group 2: AVG=41.4, STD=10.56).

Task duration vs Pre-test questionnaire (H5)

As expected, participants who demonstrated a better understanding of the introduced concepts in the pre-test questionnaire completed the main tasks (Tasks 3-8) on average faster (Fig. 7.6). Users scored between 4.3-6.9 out of 7 points (AVG=5.8, SD=0.91), with task completion times between 22-60 minutes (AVG=41.2, SD=9.07). 'non-CS' users scored above average points (AVG=6.23, STD=0.55) and completed the fastest (AVG=36.6, STD=7.46). As an outlier we observed that the fastest participant scored only 4.7 points, but easily learned how to use the system and completed the tasks in 22 minutes. Even though Group 1 scored more points in the pre-test (AVG=7.5, STD=1.35) than Group 2 (AVG=6.4, STD=2.01), both completion times were on average similar (41min).

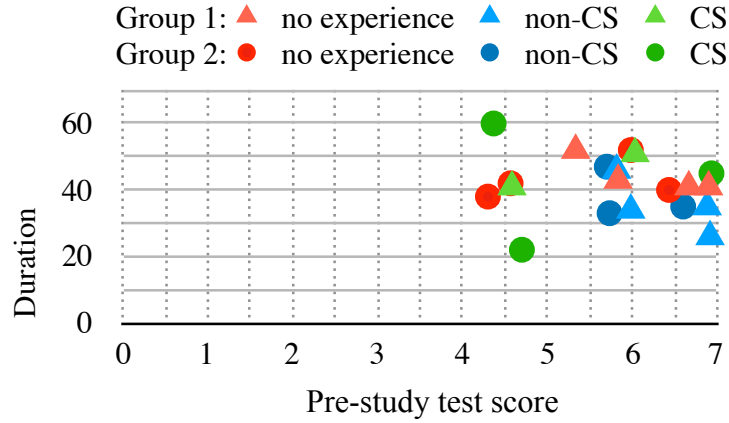


Figure 7.6: Participants who did better in the pre-test questionnaire completed the main tasks faster, with ‘non-CS’ users scoring the highest and being the fastest on average.

System usability and learnability

There are several ways to interpret the System Usability Scale (SUS) scores (Brooke [2013]) obtained from the post-study survey. Using Bangor et al. [2008] categories, 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 (Sauro [2012]), 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 which has been shown to be correlated with its learnability (Borsci et al. [2009], Sauro [2011]).

User experience

We compared the post-study questionnaire responses with those obtained in the pre-experiment user study (N=11) in Sec. 5.3, where the Wizard-of-Oz technique was used to simulate the robot programming process (Fig. 7.7). The main difference was regarding difficulties encountered during the experiment. While in Section 5.3, we had 11 (or 100%) state that they encountered no difficulties, only 7 (or 35%) of users in the last study 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 the demonstration, while in the previous study, an average of 2 (18%) disagreed. Both differences can be explained by the fact that in our latest study, users had to use an end-to-end system to program the robot, while we previously simulated the system functionalities using the Wizard-of-Oz technique. 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 this study to the question if ‘no programming experience was required’ where 13/20 (65%) agreed and 4 disagreed. While in the previous study,

9 (or 81%) agreed and only 2 (or 18%) disagreed. Overall, our user study with the working iRoPro implementation received positive responses similar to the pre-experiment user study.

Users stated the most useful feature as ‘generate solutions to defined goals automatically’ (9 or 45%), followed by ‘robot learns the action from my demonstration’ (4 or 20%) – two main aspects of our approach. A common feedback was that ‘it takes time to understand how the system works at the start’. 4 (20%) stated that the most difficult part was ‘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 a variety of different aspects. The most liked actions were ‘executing the generated plan’ (8 or 40%) and ‘demonstrating an action on Baxter’ (7 or 35%).

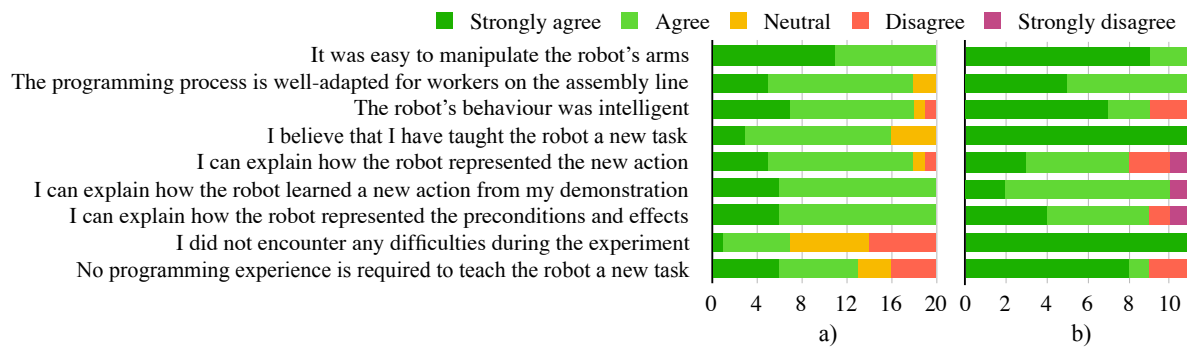


Figure 7.7: User responses from the post-study questionnaires comparing a) the iRoPro user evaluation (N=20) with responses obtained in b) the pre-experiment user study (N=11).

7.5.4 Continuous Improvement of the System

The system underwent four phases of improvement, allowing us to refine the system functionalities, graphical user interface, and user instruction methods. The four phases consisted of the following:

1. Feedback from an expert in experimental evaluation of systems that involve human-machine interaction (Dr. Nadine Mandran):

Based on the feedback we received on our initial prototype, we changed the flow for introducing the system to a novice user as it included a lot of new information (*e.g.*, Programming by demonstration and Automated planning concepts) that they were not familiar with. We also updated the questions in the pre-test questionnaire to include the same keywords (*i.e.*, predicate names) as used during the experiment to maintain consistency and allow participants to familiarise themselves.

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. 7.8) and eyes that followed the robot's moving joint (Gaisne [2018]) on the robot's screen so that it seemed more human-like. We also 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.

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.



Figure 7.8: Baxter icons used for the graphical interface (Freedman [2012]).

7.6 Discussions

Both system and user evaluations 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 in Chapter 6. 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. In the following we discuss limitations and interesting extensions of our work:

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 (Sec. 7.2.2) 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 (Li and Berenson [2016]).

7.7 Conclusion

In this chapter we presented the implementation of 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 complex tasks that go beyond 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 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 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.

Chapter 8

Conclusion and Future Work

Contents

8.1 Contributions	112
8.1.1 End-user robot programming framework	112
8.1.2 Experimental findings	112
8.1.3 Goal-oriented robot programming	113
8.1.4 iRoPro - System implementation	113
8.1.5 User and system evaluation	113
8.2 Limitations	114
8.3 Future work	115

This thesis is motivated by the idea of enabling non-robotics end-users teach robots new actions from scratch that can be generalised and reused for more complex and previously unseen tasks. We proposed an end-user robot programming framework that combines solutions from PbD and Automated Planning (Chapter 4). Unlike existing approaches where a complete task execution is taught to reach a goal, our method lets users teach the robot primitive actions and delegate the logical reasoning process of finding a solution to a task planner.

We claimed that users with or without experience in programming or related Computer Science fields can easily learn and use Automated Planning concepts needed for our framework. We validated this claim by conducting qualitative user experiments (Chapter 5). We chose Programming by Demonstration, in particular keyframe-based PbD, as it provides a good middle ground between programming difficulty for the user and data and time required to teach the robot a new skill (Sec. 2.4). We showed the generalisability and expressiveness of a goal-oriented programming approach to be a feasible end-user programming solution by addressing robotic organisation tasks (Chapter 6). Based on the obtained results, we implemented an end-user robot programming framework to teach low- and high-level actions by demonstration that can be reused with a task planner and conducted both system and user evaluations (Chapter 7).

We found that the majority of users learned how to use and navigate the system in less than an hour of training, despite being introduced to the automated planning concepts for the first time. In the following we give an overview of the contributions, discuss limitations, potential future work to address them, and perspectives for future research directions.

8.1 Contributions

8.1.1 End-user robot programming framework

In Chapter 4, we proposed iRoPro, an end-user robot programming framework that combines PbD and Automated Planning, where the robot learns action models by demonstration, and the problem of finding an action sequence is delegated to a planner. The robot programming process consists of the following steps:

1. The user teaches the robot new actions by kinesthetic demonstration, including both low- and high-level action representations.
2. The robot uses these actions with a task planner to generate solutions to user-defined problems.
3. The user can revisit the taught action models via the graphical interface to refine them.

8.1.2 Experimental findings

In Chapter 5, we conducted qualitative experiments and obtained initial results of the framework's usability and the user's ease to learn Automated Planning concepts.

- User study 1 - Findings on user acceptance of Automated Planning concepts: We observed the issues encountered when non-robotics expert users are introduced to Automated Planning concepts and asked to use the presented language to describe the world state to a robot. We evaluated the user's ability to construct symbolic action models, in terms of preconditions and effects, used by automated planners and showed that users with little to no programming experience can easily learn and use symbolic planning languages.
- User study 2 - Findings on user acceptance of the proposed robot programming framework: Using the Wizard-of-Oz technique, users were tasked to teach a robot actions by kinesthetically manipulating the robot's arm and assign action conditions that can be used for automated planning. We obtained qualitative results on user experience during the programming process as well as difficulties encountered, which contributed to the design of the iRoPro system implementation.

8.1.3 Goal-oriented robot programming

In Chapter 6, we presented work on a goal-oriented programming system for teaching robots shelf organisation tasks using keyframe-based PbD. The system allows the robot to learn low-level actions from kinesthetic demonstrations and other user input via a graphical interface. The robot learns both *what* and *how* to perform a task using PbD and a goal inference model for inferring likely shelf arrangements. We evaluated user teaching strategies with experiments on Amazon Mechanical Turk and compared their performance to eight benchmark strategies. This work focused on shelf organisation tasks with pick-and-place actions and demonstrated the representational power of a goal-oriented PbD system for end-users. The same system was used as a foundation for the implementation of the proposed robot programming framework.

8.1.4 iRoPro - System implementation

In Chapter 7, we presented the implementation of iRoPro on a Baxter research robot that

1. allows simultaneous teaching of low- and high-level actions from a single demonstration,
2. includes a user interface for action creation with condition inference and modification, and
3. allows creating and solving previously unseen problems using a task planner for the robot to execute in real-time.

The implementation includes a graphical interface, which allows the user to teach new actions by kinesthetic demonstration, modify action conditions, define new problems, and have the robot autonomously solve and execute the plan in real-time. Thus, it provides end-users with a goal-oriented approach to program robots from scratch, without writing code, therefore maximising the generalisability of taught actions with minimum programming effort.

8.1.5 User and system evaluation

We demonstrated iRoPro's capability to generalise primitive actions on six benchmark tasks that were programmed and executed on the Baxter robot (Sec. 7.4). We empirically investigated the usability of our system and validated its intuitiveness through a study with 21 users of different educational backgrounds and programming levels (Sec. 7.5). To better understand user teaching strategies, we split participants into two control groups, with and without automatic condition inference. We showed that users in both control groups can easily learn and use the system, regardless of their programming experience.

8.2 Limitations

As the developed system served as a proof-of-concept it remains an initial working prototype with several limitations. We categorise them into the main aspects of the end-to-end system and discuss possible technical improvements:

- **Improved Perception System:** An improved perception system by using multiple sensors and state-of-the-art computer vision solutions, would allow automatic detection of object types, properties and world states. This would further increase the user experience with the detection of initial states of stacked objects, automatic detection of goal states, verifying action executions, or real-time tracking of the world states. While we use PCL for object pose estimation, latest state-of-the-art techniques can estimate 6-DoF poses from RGB image data (Tremblay et al. [2018b]), allow the detection of more accurate object orientations.
- **Low-level Action Learning - Motion Planning:** While we chose keyframe-based PbD to learn low-level manipulation actions, there exist other state-of-the-art solutions such as Dynamic Movement Primitives (Pastor et al. [2009]) or the use of statistical methods for the generalisation of learned trajectories from multiple trajectories (Billard et al. [2008]). Another approach to learn primitive actions would be to segment demonstrated task executions into smaller primitive actions (Kuniyoshi et al. [1994], Wu and Demiris [2010]) that can be reused for new tasks. Furthermore, the current implementation does not take into account dynamic environments as objects are not tracked in real-time, nor problems with navigation in cluttered environments to avoid obstacles that were not present during the demonstration. In cobotic environments it is important for taught low-level actions to take into account the safety of human operators. Another extension would be to include a better motion and task planning system in order to allow executing both arms simultaneously while avoiding self-collision.
- **High-level Action Learning - Symbolic Action Representations:** We only included a minimal set of predicates (Sec. 7.2.2) that we deemed intuitive and useful for object manipulation tasks. Further user studies could involve more challenging use cases and planning domains by including a wider range of predicates. It could be interesting to infer predicates to capture more complex domains, such as object orientation (Li and Berenson [2016]) or spatial relations between objects (Tremblay et al. [2018a]). Functionality to learn the colours and shapes of new object types would increase the possible end-user applications. A possible extension would be to incorporate probabilistic techniques to learn predicates or pre-train the robot on simulated scenarios to improve the condition inference.

- **Improved Robot Programming Process:** While the overall robot programming process seems intuitive for end-users, the programming process can be improved with state-of-the-art human-computer interaction paradigms for the design of the graphical interface. The PbD process can be facilitated using Natural Language Processing solutions to allow voice commands to save poses and change gripper states. To reduce the programming overhead, it would be useful to include a feature that compares newly created actions with already existing actions to avoid redundancies.

8.3 Future work

- A known PbD problem exists with regards to the type and quality of the demonstration which is dependent on the teacher's knowledge of the robot's system. Naïve teachers often have greater assumptions on the robot's intelligence and take less care in executing demonstrations as compared to roboticists, who understand the effects of noisy demonstrations (Suay et al. [2012]). Chen and Zelinsky [2003] and Kaiser et al. [1995] recognised different sources for sub-optimality in demonstration, such as the user demonstrating unnecessary or incorrect actions due to the lack of knowledge about the task. Instructional materials in the form of tutorials and videos can be used to support the learnability of a PbD system (Cakmak and Takayama [2014]). Future research could explore the development of a standardised human-robot interaction protocol for end-user robot programming that follows successful curricula in modern education systems.
- Instead of training users directly on the real robot, an alternative approach would be to let them perform the training in a simulated environment, such as a robot simulation on the PC. This allows the user to familiarise themselves with the programming process, the involved concepts, and the graphical interface. Evaluation and testing would be less time-consuming and less risky as simulations can be restarted easily and do not involve real-life objects. Furthermore, users might be less intimidated and take more initiatives to try out different approaches to program a task for the robot.
- The proposed robot programming process involves interacting with the robot as well as with a graphical interface on a tablet or a PC. While there has been increasing research in human-robot interaction, little attention has been given to end-user robot programming. Similarly, as end-user programming solutions mostly focuses on developing programs without real robots and research in human-computer interaction addresses graphical interfaces on smartphones, tablets or PCs, there has been little work on improving the robot programming experience for end-users. Future research could explore the intersection of these domains involving end-user robot programming interfaces that include multiple interaction modalities and the robot as an additional interface.

Bibliography

- Aarup, M., Arentoft, M., Parrod, Y., Stader, J., Stokes, I., and Vadon, H. (1992). *OPTIMUM-AIV: a knowledge-based planning and scheduling system for spacecraft AIV*. University of Edinburgh, Artificial Intelligence Applications Institute.
- Abdeetedal, M. (2017). An integration platform for multiple peripheral modules with kuka robots. <https://github.com/mahyaret/KUI>. Accessed: 2019-03-12.
- Abdo, N., Kretschmar, H., Spinello, L., and Stachniss, C. (2013). Learning manipulation actions from a few demonstrations. *International Conference on Robotics and Automation*, pages 1268–1275.
- Abdo, N., Stachniss, C., Spinello, L., and Burgard, W. (2015). Robot, organize my shelves! Tidying up objects by predicting user preferences. *International Conference on Robotics and Automation*, pages 1557–1564.
- Active8 Robots, O. (2014). Baxter research robot - providing support for the ebola outbreak. <http://www.active8robots.com/news/baxter-ebola-research>. Accessed: 2019-03-12.
- Ahmadzadeh, S. R., Kormushev, P., and Caldwell, D. G. (2013). Visuospatial skill learning for object reconfiguration tasks. *International Conference on Intelligent Robots and Systems*, pages 685–691.
- Ahmadzadeh, S. R., Paikan, A., Mastrogiovanni, F., Natale, L., Kormushev, P., and Caldwell, D. G. (2015). Learning symbolic representations of actions from human demonstrations. *International Conference on Robotics and Automation*, pages 3801–3808.
- Akgun, B., Cakmak, M., Jiang, K., and Thomaz, A. L. (2012). Keyframe-based learning from demonstration. *International Journal of Social Robotics*, pages 343–355.
- Akgun, B. and Subramanian, K. (2011). Robot learning from demonstration: kinesthetic teaching vs. teleoperation. *Unpublished manuscript*.

- Akgun, B. and Thomaz, A. (2016). Simultaneously learning actions and goals from demonstration. *Autonomous Robots*, 40(2):211–227.
- Alexandrova, S., Cakmak, M., Hsiao, K., and Takayama, L. (2014). Robot Programming by Demonstration with Interactive Action Visualizations. *Robotics: Science and Systems*.
- Alexandrova, S., Tatlock, Z., and Cakmak, M. (2015). RoboFlow: A flow-based visual programming language for mobile manipulation tasks. *International Conference on Robotics and Automation*, pages 5537–5544.
- Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483.
- Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*.
- Avenier, M.-J. and Thomas, C. (2015). Finding one’s way around various methodological guidelines for doing rigorous case studies: A comparison of four epistemological frameworks. In *Systèmes d’information & management*, pages vol. 20, no. 1, pp. 61–98.
- Backes, P. G., Norris, J. S., Powell, M. W., and Vona, M. A. (2004). Multi-mission activity planning for mars lander and rover missions. In *Aerospace Conference*, volume 2, pages 877–886. IEEE.
- Bangor, A., Kortum, P. T., and Miller, J. T. (2008). An empirical evaluation of the system usability scale. *International Journal of Human-Computer Interaction*, 24(6):574–594.
- Barakova, E. I., Gillesen, J. C., Huskens, B. E., and Lourens, T. (2013). End-user programming architecture facilitates the uptake of robots in social therapies. *Robotics and Autonomous Systems*, 61(7):704–713.
- Barták, R. and Skalic̃ký, T. (2009). A local approach to automated correction of violated precedence and resource constraints in manually altered schedules. *Multidisciplinary International Scheduling Conference: Theory and Applications*.
- Biggs, G. and Macdonald, B. (2003). A Survey of Robot Programming Systems. *Australasian Conference on Robotics and Automation*, pages 1–3.
- Billard, A., Calinon, S., Dillmann, R., and Schaal, S. (2008). Robot programming by demonstration. In *Springer Handbook of Robotics*, pages 1371–1394. Springer.
- Billard, A. G., Calinon, S., and Dillmann, R. (2016). Learning from humans. In *Springer handbook of robotics*, pages 1995–2014. Springer.

- Billing, E. A. and Hellström, T. (2010). A Formalism for Learning from Demonstration. *Paladyn, Journal of Behavioral Robotics*, 1(1):1–13.
- Bischoff, R., Kazi, A., and Seyfarth, M. (2002). The MORPHA style de for icon-based programming. *International Symposium on Robot and Human Interactive Communication*, pages 482–487.
- Blum, A. and Furst, M. (1997). Fast Planning Through Planning Graph Analysis. *Artificial Intelligence*, 90(1-2):281–300.
- Bonet, B. and Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33.
- Borsci, S., Federici, S., and Lauriola, M. (2009). On the dimensionality of the system usability scale: a test of alternative measurement models. *Cognitive processing*, 10(3):193–197.
- Braumann, J. and Brell-Cokcan, S. (2011). Parametric robot control: integrated CAD/CAM for architectural design. In *Conference of the Association for Computer Aided Design in Architecture*, pages 242–251.
- Brenner, M. (2003). A multiagent planning language. In *International Conference on Automated Planning and Scheduling*, pages 33–38.
- Bresina, J. L., Jónsson, A. K., Morris, P. H., and Rajan, K. (2005). Activity Planning for the Mars Exploration Rovers. *International Conference on Automated Planning and Scheduling*, pages 40–49.
- Brooke, J. (1996). SUS - A quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7.
- Brooke, J. (2013). SUS: A retrospective. *Journal of usability studies*, 8(2):29–40.
- Cakmak, M. and Lopes, M. (2012). Algorithmic and Human Teaching of Sequential Decision Tasks. In *Association for the Advancement of Artificial Intelligence*.
- Cakmak, M. and Takayama, L. (2014). Teaching people how to teach robots: The effect of instructional materials and dialog design. *International Conference on Human-Robot Interaction*, pages 431–438.
- Calinon, S. and Billard, A. (2007a). Active teaching in robot programming by demonstration. *International Symposium on Robot and Human Interactive Communication*, pages 702–707.
- Calinon, S. and Billard, A. (2007b). Incremental learning of gestures by imitation in a humanoid robot. *International Conference on Human-Robot Interaction*, pages 255–262.

- Calinon, S. and Billard, A. (2009). Statistical learning by imitation of competing constraints in joint and task space. *Advanced Robotics*, 23(15):2059–2076.
- Cashmore, M. and Fox, M. (2015). ROSPlan: Planning in the Robot Operating System. In *International Conference on Automated Planning and Scheduling*, pages 333–341.
- Chapman, D. (1987). Planning for conjunctive goals. *Artificial intelligence*, 32(3):333–377.
- Chen, J. and Zelinsky, A. (2003). Programing by demonstration: Coping with suboptimal teaching actions. *The International Journal of Robotics Research*, 22(5):299–319.
- Chernova, S. and Thomaz, A. L. (2014). Robot learning from human teachers. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 8(3):1–121.
- Clapaud, A. (2013). Des cobots rb3d pour le montage des airbus a350 en 2015. <http://www.4erevolution.com/en/cobots-rb3d-airbus-a350/>. Accessed: 2019-03-12.
- Colgate, J. and Peshkin, M. (1999). Cobots. Google Patents. US Patent 5,952,796.
- Cooper, M. C., de Roquemaurel, M., and Régnier, P. (2011). A weighted CSP approach to cost-optimal planning. *AI Communications*, 24(1):1–29.
- Correll, N., Bekris, K. E., Berenson, D., Brock, O., Causo, A., Hauser, K., Okada, K., Rodriguez, A., Romano, J. M., and Wurman, P. R. (2016). Analysis and observations from the first amazon picking challenge. *Transactions on Automation Science and Engineering*.
- Cresswell, S. N., McCluskey, T. L., and West, M. M. (2013). Acquiring planning domain models using LOCM. *The Knowledge Engineering Review*, 28(2):195–213.
- Crowe, S. (2018). Rethink Robotics IP acquired by HAHN Group. <https://www.therobotreport.com/hahn-group-acquires-rethink-robotics-ip>. Accessed: 2019-03-28.
- Cubek, R., Ertel, W., and Palm, G. (2015). High-level learning from demonstration with conceptual spaces and subspace clustering. In *International Conference on Robotics and Automation*, pages 2592–2597. IEEE.
- Di Rocco, M., Pecora, F., Sathyakeerthy, S., Grosinger, J., Saffiotti, A., Bonaccorsi, M., Limosani, R., Manzi, A., Cavallo, F., Dario, P., et al. (2014). A planner for ambient assisted living: From high-level reasoning to low-level robot execution and back. In *AAAI Spring symposium series*.

- Do, M. and Kambhampati, S. (2001). Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artificial Intelligence*, 132(2):151–182.
- Dogar, M. R. and Srinivasa, S. S. (2010). Push-grasping with dexterous hands: Mechanics and a method. *International Conference on Intelligent Robots and Systems*, pages 2123–2130.
- Dvorak, F., Bit-Monnot, A., Ingrand, F., and Ghallab, M. (2014a). A flexible ANML actor and planner in robotics. In *ICAPS Planning and Robotics (PlanRob) Workshop*.
- Dvorak, F., Bit-Monnot, A., Ingrand, F., Ghallab, M., et al. (2014b). Plan-Space Hierarchical Planning with the Action Notation Modeling Language. *International Conference on Tools with Artificial Intelligence*.
- Ekvall, S. and Kragic, D. (2008). Robot learning from demonstration: a task-level planning approach. *International Journal of Advanced Robotic Systems*, 5(3):223–234.
- Faggella, D. (2019). Machine learning in robotics – 5 modern applications. <https://emerj.com/ai-sector-overviews/machine-learning-in-robotics/>. Accessed: 2019-04-15.
- Fernández, E., Crespo, L. S., Mahtani, A., and Martinez, A. (2015). *Learning ROS for Robotics Programming*. Packt Publishing Ltd.
- Fernández, S., Adarve, R., Pérez, M., Rybarczyk, M., and Borrajo, D. (2006). Planning for an AI based virtual agents game. *International Conference on Automated Planning and Scheduling*, page 14.
- Ferrer-Mestres, J., Frances, G., and Geffner, H. (2015). Planning with state constraints and its application to combined task and motion planning. In *ICAPS Planning and Robotics (PlanRob) Workshop*.
- Fikes, R. E. and Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208.
- Fischer, K., Kirstein, F., Jensen, L. C., Krüger, N., Kukliński, K., Savarimuthu, T. R., et al. (2016). A comparison of types of robot control for programming by demonstration. In *International Conference on Human-Robot Interaction*, pages 213–220. Press.
- Forbes, M., Rao, R. P., Zettlemoyer, L., and Cakmak, M. (2015). Robot programming by demonstration with situated spatial language understanding. In *International Conference on Robotics and Automation*, pages 2014–2020. IEEE.
- Fox, M. and Long, D. (2003). PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research*, pages 61–124.

- Fraser, N. (2013). Blockly: A visual programming editor. <https://developers.google.com/blockly/>. Accessed: 2019-04-01.
- Freedman, D. H. (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: 2019-04-11.
- Gaisne, A. (2018). Baxter eyes. https://github.com/Anne-Gaisne/baxter_eyes. Accessed: 2019-04-11.
- Garrett, C. R., Lozano-Pérez, T., and Kaelbling, L. P. (2015). Ffrob: An efficient heuristic for task and motion planning. In *Algorithmic Foundations of Robotics XI*, pages 179–195. Springer.
- Ghallab, M., Knoblock, C., Wilkins, D., Barrett, A., Christianson, D., Friedman, M., Kwok, C., Golden, K., Penberthy, S., Smith, D., Sun, Y., and Weld, D. (1998). PDDL - The Planning Domain Definition Language. *International Planning Competition Committee*.
- Ghallab, M., Nau, D., and Traverso, P. (2004). *Automated planning: theory & practice*. Elsevier.
- github.com/Elucidation (2012). N-disk Hanoi PDDL generator. <https://github.com/Elucidation/Hanoi-PDDL-generator>. Accessed: 2019-03-12.
- github.com/pellierd (2017). PDDL4J ROSPY. https://github.com/pellierd/pddl4j_rospy. Accessed: 2019-03-18.
- github.com/phuicy (2014). Downward : A ROS implementation of the Fast Downward Planner. <https://github.com/phuicy/downward>. Accessed: 2019-03-18.
- Glas, D., Satake, S., Kanda, T., and Hagita, N. (2012). An interaction design framework for social robots. In *Robotics: Science and Systems*, volume 7, page 89.
- Glas, D. F., Kanda, T., and Ishiguro, H. (2016). Human-robot interaction design using Interaction Composer: Eight years of lessons learned. In *International Conference on Human-Robot Interaction*, pages 303–310. Press.
- Glinský, R. and Barták, R. (2011). VisPlan—interactive visualisation and verification of plans. *Knowledge Engineering for Planning and Scheduling*, pages 134–138.
- Gosavi, A. (2009). Reinforcement learning: A tutorial survey and recent advances. *Journal on Computing*, 21(2):178–192.

- Grollman, D. H. and Billard, A. G. (2012). Robot learning from failed demonstrations. *International Journal of Social Robotics*, 4(4):331–342.
- Helmert, M. (2006). The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246.
- Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Sendonaris, A., Dulac-Arnold, G., Osband, I., Agapiou, J., et al. (2017). Learning from demonstrations for real world reinforcement learning. *arXiv preprint arXiv:1704.03732*.
- Hoffmann, J. and Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302.
- Hofstadter, D. R. (1985). *Metamagical themas: Questing for the essence of mind and pattern*. Basic books.
- Huang, J. (2018a). Rapid PbD. https://github.com/jstnhuang/rapid_pbd. Accessed: 2019-04-11.
- Huang, J. (2018b). Surface perception. https://github.com/jstnhuang/surface_perception. Accessed: 2019-04-11.
- Huang, J. and Cakmak, M. (2017). Code3: A system for end-to-end programming of mobile manipulator robots for novices and experts. In *International Conference on Human-Robot Interaction*, pages 453–462. ACM.
- Huang, J., Lau, T., and Cakmak, M. (2016). Design and evaluation of a rapid programming system for service robots. In *International Conference on Human-Robot Interaction*, pages 295–302. Press.
- Huckaby, J., Vassos, S., and Christensen, H. I. (2013). Planning with a task modeling framework in manufacturing robotics. In *International Conference on Intelligent Robots and Systems*, pages 5787–5794. IEEE.
- Hudak, P., Courtney, A., Nilsson, H., and Peterson, J. (2002). Arrows, robots, and functional reactive programming. In *International School on Advanced Functional Programming*, pages 159–187. Springer.
- Jansen, B. and Belpaeme, T. (2006). A model for inferring the intention in imitation tasks. In *International Symposium on Robot and Human Interactive Communication*, pages 238–243. IEEE.
- Jetchev, N., Lang, T., and Toussaint, M. (2013). Learning grounded relational symbols from continuous data for abstract reasoning.

- Jilani, R., Crampton, A., Kitchin, D. E., and Vallati, M. (2014). Automated knowledge engineering tools in planning: state-of-the-art and future challenges. *ICAPS Knowledge Engineering for Planning and Scheduling*.
- Jones, T. (2017). Models for machine learning. <https://www.ibm.com/developerworks/library/cc-models-machine-learning/index.html>. Accessed: 2018-07-08.
- Jund, P., Abdo, N., Eitel, A., and Burgard, W. (2016). The Freiburg Groceries Dataset. *arXiv preprint arXiv:1611.05799*.
- Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.
- Kaiser, M., Friedrich, H., and Dillmann, R. (1995). Obtaining good performance from a bad teacher. In *Programming by Demonstration vs. Learning from Examples Workshop at ML*, volume 95.
- Kautz, H. and Selman, B. (1999). Unifying SAT-based and Graph-based Planning. In *International Joint Conference on Artificial Intelligence*, pages 318–325.
- Kautz, H., Selman, B., and Hoffmann, J. (2006). SATPlan: Planning as Satisfiability. In *Abstracts of the 5th International Planning Competition*.
- Khan, F., Mutlu, B., and Zhu, X. (2011). How do humans teach: On curriculum learning and teaching dimension. In *Advances in Neural Information Processing Systems*, pages 1449–1457.
- King, J. E., Haustein, J. A., Srinivasa, S. S., and Asfour, T. (2015). Nonprehensile whole arm rearrangement planning on physics manifolds. In *International Conference on Robotics and Automation*, pages 2508–2515. IEEE.
- Kittur, A., Chi, E. H., and Suh, B. (2008). Crowdsourcing User Studies with Mechanical Turk. In *Conference on Human Factors in Computing Systems*, pages 453–456. ACM.
- Kober, J., Bagnell, J. A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274.
- Kolobov, A. (2012). Planning with Markov decision processes: An AI perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–210.
- Konidaris, G., Kaelbling, L. P., and Lozano-Perez, T. (2018). From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289.

- Kootbally, Z., Schlenoff, C., Lawler, C., Kramer, T., and Gupta, S. K. (2015). Towards robust assembly with knowledge representation for the planning domain definition language (PDDL). *Robotics and Computer-Integrated Manufacturing*, 33:42–55.
- Kormushev, P., Calinon, S., and Caldwell, D. G. (2013). Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2(3):122–148.
- Kuhner, D., Aldinger, J., Burget, F., Göbelbecker, M., Burgard, W., and Nebel, B. (2018). Closed-loop robot task planning based on referring expressions. In *International Conference on Intelligent Robots and Systems*, pages 876–881. IEEE.
- Kuniyoshi, Y., Inaba, M., and Inoue, H. (1994). Learning by watching: Extracting reusable task knowledge from visual observation of human performance. 10(6):799–822.
- Kurenkov, A., Akgun, B., and Thomaz, A. L. (2015). An evaluation of GUI and kinesthetic teaching methods for constrained-keyframe skills. *International Conference on Intelligent Robots and Systems*.
- Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., Salvucci, D., Scheutz, M., Thomaz, A., Trafton, G., et al. (2017). Interactive task learning. *Intelligent Systems*, 32(4):6–21.
- Lego (2003). Lego mindstorms. <http://mindstorms.lego.com>. Accessed: 2019-03-12.
- Li, C. and Berenson, D. (2016). Learning object orientation constraints and guiding constraints for narrow passages from one demonstration. In *International symposium on experimental robotics*, pages 197–210. Springer.
- Li, C., Yang, C., Wan, J., Annamalai, A., and Cangelosi, A. (2017). Neural learning and kalman filtering enhanced teaching by demonstration for a baxter robot. In *International Conference on Automation and Computing*, pages 1–6.
- Lipovetzky, N. and Geffner, H. (2017). Best-first width search: Exploration and exploitation in classical planning. In *AAAI*.
- Long, D. and Fox, M. (1999). Efficient implementation of the plan graph in stan. *Journal of Artificial Intelligence Research*, 10:87–115.
- Lopez, A. and Bacchus, F. (2003). Generalizing graphplan by formulating planning as a CSP. In *International Conference on Artificial Intelligence*, pages 954–960.
- Lozano-Perez, T. (1983). Robot programming. *Proceedings of the IEEE*, 71(7):821–841.

- Majed, M. (2014). *Learn to program with Scratch: a visual introduction to programming with games, art, science, and math*. No Starch Press Inc.
- Mandran, N. (2018). Traceable Human Experiment Design Research: Theoretical Model and Practical Guide. In *ISTE ltd*.
- Mandran, N. and Dupuy-Chessa, S. (2017). THEDRE: a Traceable Process for High Quality in Human Centred Computer Science Research. In *International conference of System Development*.
- Martínez, D., Alenya, G., and Torras, C. (2017). Relational reinforcement learning with guided demonstrations. *Artificial Intelligence*, 247:295–312.
- Mason, M. and Lopes, M. C. (2011). Robot self-initiative and personalization by learning through repeated interactions. *International Conference on Human-Robot Interaction*, pages 433–440.
- McAllester, D. and Rosenblatt, D. (1991). Systematic nonlinear planning. *AAAI*.
- McCluskey, T. and Simpson, R. (2005). Using GIPO to support learning in knowledge acquisition and automated planning.
- McCluskey, T. L., Cresswell, S., Richardson, N. E., and West, M. M. (2009). Automated acquisition of action knowledge. *International Joint Conference on Agents and Artificial Intelligence*.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.
- Mohseni-Kabir, A., Rich, C., Chernova, S., Sidner, C. L., and Miller, D. (2015). Interactive hierarchical task learning from a single demonstration. *International Conference on Human-Robot Interaction*, pages 205–212.
- Mollard, Y., Munzer, T., Baisero, A., Toussaint, M., and Lopes, M. (2015). Robot programming from demonstration, feedback and transfer. *International Conference on Intelligent Robots and Systems*, pages 1825–1831.
- Mühe, H., Angerer, A., Hoffmann, A., and Reif, W. (2010). On reverse-engineering the kuka robot language. *arXiv preprint arXiv:1009.5004*.
- Muise, C. (2016). Planning. domains. *International Conference on Automated Planning and Scheduling: System Demonstration*.

- Nau, D. S. (2007). Current trends in automated planning. *AI magazine*, 28(4):43–43.
- Nehaniv, C. L., Dautenhahn, K., et al. (2002). The correspondence problem. *Imitation in animals and artifacts*.
- Nguyen, H., Ciocarlie, M., Hsiao, K., and Kemp, C. C. (2013). Ros commander (rosco): Behavior creation for home robots. *International Conference on Robotics and Automation*, pages 467–474.
- Niculescu, M. N. and Mataric, M. J. (2003). Natural methods for robot task learning: Instructive demonstrations, generalization and practice. *International Joint Conference on Autonomous agents and multiagent systems*, pages 241–248.
- Niekum, S., Chitta, S., Barto, A., Marthi, B., and Osentoski, S. (2013). Incremental semantically grounded learning from demonstration. *Robotics: Science and Systems*, 9.
- Niekum, S., Osentoski, S., Konidaris, G., and Barto, A. G. (2012). Learning and generalization of complex tasks from unstructured demonstrations. In *International Conference on Intelligent Robots and Systems*, pages 5239–5246. IEEE.
- Orendt, E. M., Fichtner, M., and Henrich, D. (2016). Robot programming by non-experts: intuitiveness and robustness of one-shot robot programming. *International Symposium on Robot and Human Interactive Communication*, pages 192–199.
- Pardowitz, M., Knoop, S., Dillmann, R., and Zollner, R. (2007). Incremental learning of tasks from user demonstrations, past experiences, and vocal comments. *Transactions on Systems, Man, and Cybernetics*, 37(2):322–332.
- Pastor, P., Hoffmann, H., Asfour, T., and Schaal, S. (2009). Learning and generalization of motor skills by learning from demonstration. *International Conference on Robotics and Automation*, pages 763–768.
- Paxton, C., Hundt, A., Jonathan, F., Guerin, K., and Hager, G. D. (2017). CoSTAR: Instructing collaborative robots with behavior trees and vision. *International Conference on Robotics and Automation*, pages 564–571.
- Pellier, D. and Fiorino, H. (2018). Pddl4j: a planning domain description library for java. *Journal of Experimental & Theoretical Artificial Intelligence*, 30(1):143–176.
- Penberthy, J. S., Weld, D. S., et al. (1992). UCPOP: A Sound, Complete, Partial Order Planner for ADL. *International Conference on Principles of Knowledge Representation and Reasoning*, pages 103–114.

- Peppoloni, L., Di Fava, A., Ruffaldi, E., and Avizzano, C. A. (2014). A ROS-integrated architecture to learn manipulation tasks from a single demonstration. *International Symposium on Robot and Human Interactive Communication*, pages 537–542.
- Perzylo, A., Somani, N., Profanter, S., Kessler, I., Rickert, M., and Knoll, A. (2016). Intuitive instruction of industrial robots: Semantic process descriptions for small lot production. *International Conference on Intelligent Robots and Systems*, pages 2293–2300.
- Plch, T., Chomut, M., Brom, C., and Barták, R. (2012). Inspect, edit and debug PDDL documents: Simply and efficiently with PDDL studio. *International Conference on Automated Planning and Scheduling: System Demonstration*.
- Profanter, S., Perzylo, A., Somani, N., Rickert, M., and Knoll, A. (2015). Analysis and semantic modeling of modality preferences in industrial human-robot interaction. In *International Conference on Intelligent Robots and Systems*, pages 1812–1818. IEEE.
- Quigley, M., Faust, J., Foote, T., and Leibs, J. ROS: an open-source Robot Operating System.
- Ramirez, M., Lipovetzky, N., and Muise, C. (2015). Lightweight Automated Planning ToolKiT. <http://lapkt.org/>. Accessed: 2019-03-12.
- Richter, S. and Westphal, M. (2010). The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177.
- Rintanen, J. (2014). Madagascar: Scalable planning with SAT. *International Planning Competition*.
- Robotic Industries Association, R. O. M. T. (2017). Robotics in Agriculture: Types and Applications. <https://www.robotics.org/blog-article.cfm/Robotics-in-Agriculture-Types-and-Applications/74>. Accessed: 2019-04-04.
- Robotic Industries Association, R. O. M. T. (2018). Pick and Place Robots: What Are They Used For and How Do They Benefit Manufacturers? <https://www.robotics.org/blog-article.cfm/Pick-and-Place-Robots-What-Are-They-Used-For-and-How-Do-They-Benefit-Manufacturers/88>. Accessed: 2019-04-11.
- Robotics, R. (2008–2016). Baxter research robot. <http://www.rethinkrobotics.com/baxter-research-robot/>. Accessed: 2016-04-21.
- Robotics, R. (2012). Inera software. <https://www.rethinkrobotics.com/intera>. Accessed: 2019-03-28.
- Robotics, R. (2014). Baxter Reseach Robot - Connect 4. http://sdk.rethinkrobotics.com/wiki/Connect_Four_Demo. Accessed: 2019-03-12.

- Sauro, J. (2011). A practical guide to the System Usability Scale: Background, benchmarks & best practices. *Measuring Usability LLC*.
- Sauro, J. (2012). Predicting net promoter scores from system usability scale scores. <https://measuringu.com/nps-sus/>. Accessed: 2019-04-16.
- Schaal, S., Peters, J., Nakanishi, J., and Ijspeert, A. (2003). Learning Movement Primitives. *International Symposium on Robotics Research*.
- Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, 61:85–117.
- Schulman, J., Ho, J., Lee, C., and Abbeel, P. (2013). Learning from demonstrations through the use of non-rigid registration. *International Journal of Robotics Research*.
- Shah, M., Chrapa, L., Jimoh, F., Kitchin, D., McCluskey, T., Parkinson, S., and Vallati, M. (2013). Knowledge engineering tools in planning: State-of-the-art and future challenges. *International Conference on Automated Planning and Scheduling*.
- She, L., Cheng, Y., Chai, J. Y., Jia, Y., Yang, S., and Xi, N. (2014). Teaching robots new actions through natural language instructions. *International Symposium on Robot and Human Interactive Communication*, pages 868–873.
- Simpson, R. M., Kitchin, D. E., and McCluskey, T. L. (2007). Planning domain definition using gipo. *The Knowledge Engineering Review*, 22(2):117–134.
- Škopek, O. and Barták, R. (2017). Transporteditor—creating and visualising transportation problems and plans. *International Conference on Automated Planning and Scheduling*.
- Slaney, J. and Thiébaux, S. (2001). Blocks world revisited. *Artificial Intelligence*, 125(1-2):119–153.
- Stager, D., Sergi-Curfman, M., LaRose, D., Fromme, C., Bagnell, J., Cuzzillo, E., and Baker, L. (2013). Computer vision and machine learning software for grading and sorting plants. US Patent App. 13/634,086.
- Stenmark, M., Haage, M., and Topp, E. A. (2017). Simplified programming of re-usable skills on a safe industrial robot: Prototype and evaluation. In *International Conference on Human-Robot Interaction*, pages 463–472. ACM.
- Suay, H. B., Toris, R., and Chernova, S. (2012). A practical comparison of three robot learning from demonstration algorithm. *International Journal of Social Robotics*, 4(4):319–330.

- Sutton, R. S. et al. (1998). *Introduction to reinforcement learning*, volume 135. Cambridge: MIT press.
- Tate, A., Drabble, B., and Kirby, R. (1994). O-Plan2: an open architecture for command, planning and control. *Intelligent Scheduling*.
- Technavio (2018). 6 Major Types of Industrial Robots Used in the Global Manufacturing 2018. <https://blog.technavio.com/blog/major-types-of-industrial-robots>. Accessed: 2019-04-15.
- Tremblay, J., To, T., Molchanov, A., Tyree, S., Kautz, J., and Birchfield, S. (2018a). Synthetically trained neural networks for learning human-readable plans from real-world demonstrations. *International Conference on Robotics and Automation*, pages 1–5.
- Tremblay, J., To, T., Sundaralingam, B., Xiang, Y., Fox, D., and Birchfield, S. (2018b). Deep object pose estimation for semantic robotic grasping of household objects. *arXiv preprint arXiv:1809.10790*.
- Triantafillou, E., Baier, J., and McIlraith, S. (2015). A unifying framework for planning with LTL and regular expressions. *ICAPS Workshop on Model-Checking and Automated Planning*, pages 23–31.
- Tullis, T. S. and Stetson, J. N. (2004). A comparison of questionnaires for assessing website usability. *Usability professional association conference*.
- Ueda, R. (2018). pddl_planner. http://docs.ros.org/indigo/api/pddl_planner. Accessed: 2019-03-18.
- Ugur, E. and Piater, J. (2015). Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning. *International Conference on Robotics and Automation*, pages 2627–2633.
- Universal Robots, O. (2016). How cobots transform the food industry. <http://www.universal-robots.com/blog/how-cobots-transform-the-food-industry/>. Accessed: 2019-03-12.
- University of Cambridge, I. D. G. (2013). Concept design process: Overview. http://www.inclusivedesign toolkit.com/GS_overview/overview.html. Accessed: 2019-04-11.
- Vallati, M., Chrupa, L., Grzes, M., McCluskey, T., Roberts, M., and Sanner, S. (2014). The 2014 International Planning Competition: Progress and Trends. *AI Magazine*.

- Vaquero, T. S., Silva, J. R., Tonidandel, F., and Beck, J. C. (2013). itSIMPLE: towards an integrated design system for real planning applications. *The Knowledge Engineering Review*, 28(2):215–230.
- Veeraraghavan, H. and Veloso, M. (2008). Teaching sequential tasks with repetition through demonstration. *International Joint Conference on Autonomous agents and multiagent systems*, pages 1357–1360.
- Vodrázka, J. and Chrpá, L. (2010). Visual design of planning domains. *International Conference on Automated Planning and Scheduling*.
- Vyvyan Pugh, A. R. (2013-2014). Baxter Reseach Robot - Worked Example Visual Servoing. http://sdk.rethinkrobotics.com/wiki/Worked_Example_Visual_Servoing. Accessed: 2019-03-12.
- Walsh, T. J. (2010). *Efficient learning of relational models for sequential decision making*. Doctoral dissertation, Rutgers University-Graduate School-New Brunswick.
- Wrede, S., Emmerich, C., Grünberg, R., Nordmann, A., Swadzba, A., and Steil, J. (2013). A user study on kinesthetic teaching of redundant robots in task and configuration space. *Journal of Human-Robot Interaction*, 2(1):56–81.
- Wu, Y. and Demiris, Y. (2010). Hierarchical learning approach for one-shot action imitation in humanoid robots. In *International Conference on Control Automation Robotics & Vision*, pages 453–458. IEEE.
- Yang, C., Liang, P., Ajoudani, A., Li, Z., and Bicchi, A. (2016). Development of a robotic teaching interface for human to human skill transfer. *International Conference on Intelligent Robots and Systems*, pages 710–716.
- Yang, Y., Li, Y., Fermüller, C., and Aloimonos, Y. (2015). Robot learning manipulation action plans by “watching” unconstrained videos from the world wide web. *Conference on Artificial Intelligence*, pages 3686–3692.
- Younes, H. and Littman, M. (2004). PPDDL 1.0: an extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, Carnegie Mellon University.

Appendix A

Publications and Submissions

Here is the list of papers that have been published or submitted during the Ph.D.:

1. Y.S. Liang, D. Pellier, H. Fiorino, S. Pesty, “End-User Programming of Low- and High-Level Actions for Robotic Task Planning”, Robotics: Science and Systems (RSS), Freiburg, Germany, 2019. (In Review)
2. Y. S. Liang, D. Pellier, H. Fiorino, S. Pesty and M. Cakmak, “Simultaneous End-User Programming of Goals and Actions for Robotic Shelf Organization”, In Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 2018, pp. 6566-6573.
3. Y.S. Liang, D. Pellier, H. Fiorino, S. Pesty, “Evaluation of a robot programming framework for non-experts using symbolic planning representations”, In Proceedings for the 2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), Lisbon, Portugal, 2017, pp. 1121-1126.
4. Y.S. Liang, D. Pellier, H. Fiorino, S. Pesty, “A Framework for Robot Programming in Cobot environments: First User Experiments”, In Proceedings of the 3rd ACM International Conference on Mechatronics and Robotics Engineering (ICMRE 2017), Paris, France, 2017, pp. 30-35.

Appendix B

Resources for Pre-Experiment: Study 1 (Sec. 5.2)

The following documents were used in the first user study. As all experiments were conducted in French, all documents are in French:

- Experimental protocol
- Post-study questionnaire
- Action schema used

Protocole d'expérimentation pour les concepts de la planification

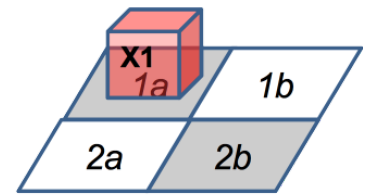
Personnes présentes

- K: expérimentateur
- S: sujet de l'expérimentation

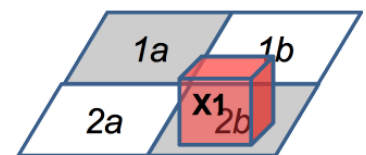
Niveau 1 – Tâche 1

Nom d'objet	Type	Propriété (état initial)	Propriété généralisé
X1	cube	est_rouge (X1)	est_rouge (cube)
		est_sur(X1,1a)	est_sur(cube,position)
1a	position	non_vide(1a)	non_vide(position)
1b	position	est_vide(1b)	est_vide(position)
2a	position	est_vide(2a)	est_vide(position)
2b	position	est_vide(2b)	est_vide(position)

État initial



État final



(Pour chaque nouvelle tâche, K déplace les objets sur le damier pour les états initiaux)

Pour qu'il fasse les déplacements soi-même, il faut qu'il comprenne les objets qui existent et qu'il peut déplacer. Donc, il faut lui expliquer les objets qui existent et que tu vois sur la table dans un langage qu'il comprend. On va faire la première exemple ensemble.

1. Echauffement: Voici un premier état initial.

K met le cube rouge sur le damier à la position 1a.

1.1 Types: - cube

- Quels sont les noms d'objet que tu vois sur le damier?
- Quel est le type de l'objet X1?

Le type est une façon de décrire l'objet X1 d'une manière plus généralisée.

1.2 Propriétés (initial): - cube

- Quelle est la propriété de l'objet X1 que tu observes dans cet état sur le damier?
 - Si S ne dit pas, K propose de regarder la couleur (ou la position)

Dans le langage que comprend le robot, on écrit

- 'est_rouge(X1)' pour dire 'X1 est rouge'
- 'est_sur(X1,1a)' pour dire 'X1 est sur 1a'

'verbe' ('sujet', 'compléments'): Le verbe est au début, puis le sujet, et à la fin il y a des compléments.

1.3 Propriétés (généralisée): - cube

- Comment peux-tu exprimer cette propriété dans une façon plus généralisée pour qu'on peut l'utiliser pour n'importe quel objet de ce type?
- K demande "qu'est-ce qui manque encore?" (position/couleur)

1.2. Types/Propriétés: - position

- Est-ce que le tableau est complet?
 - Si S dit oui, K demande "Est-ce que tu vois d'autre noms/types sur le damier?"
- Remplissez le tableau (nom, type). (K continue à guider S)
- Quelles sont leurs propriétés?
 - Si S n'y arrive pas, K guide "Quelle est la différence entre la position 1a et les autres?"
- Propriété généralisée?
 - Si S n'y arrive pas, K "Comparez-la avec les propriétés généralisées qu'on a déjà rempli"

1.3 Actions: K donne à S la fiche d'action I avec le tableau vide

Maintenant on va voir comment on peut modéliser une action. Observez les trois graphiques.

Une action consiste des préconditions, c'est-à-dire des conditions qui sont nécessaire pour effectuer l'action, et des effets, les conditions qui s'appliquent après l'action.

1.3.1: l'action dans le langage: déplace(X1,1a,2b)

- Quel est l'action qui est effectuée?
 - Si S n'y arrive pas, K "On peut dire que le cube est déplacé de 1a à 2b"
- Comment peut-on représenter cette action dans le langage? Remplissez la case 'action'.

déplace(X1,1a,2b): si S n'écrit pas toutes les paramètres X1, 1a, 2b - p.ex. déplacer(X1):

- Si on lis que l'action, est-ce qu'on peut comprendre ce qu'il faut faire? (de déplacer de 1a à 2b)
- Qu'est-ce que cette action que tu viens d'écrire veut dire en français?
- Qu'est-ce qu'il manque pour que l'action dise "déplacé X1 de 1a à 2b"?

1.3.2: préconditions: est_sur(X1,1a)

- Quelles sont les conditions nécessaire pour déplacer le cube? Observez l'état initial.

Si S donne la bonne réponse: *est_sur(X1,1a)*

- K explique: C'est exacte, pour déplacer le cube X1 de 1a à 2b, il faut que X1 soit d'abord sur 1a.

Si S donne plus de conditions que nécessaire:

- Est-ce qu'on a besoin de toutes les conditions pour déplacer le cube?
- Est-ce qu'il est nécessaire que X1 soit rouge pour déplacer le cube X1?

1.3.3: effets: est_sur(X1,2b)

- Quelles sont les effets après l'action a été effectuée?
- Qu'est-ce qu'il change après le déplacement?

1.4 Action généralisée:

De la même manière qu'on a déjà fait dans le tableau, on veut généraliser l'action.

- Que seraient l'action généralisée pour qu'on puisse utiliser cette action pour n'importe quel objet de ce type?
 - Si S n'y arrive pas, K propose "Comment on a généralisé la propriété dans le tableau?"
- Que seraient les conditions généralisées?
 - Si S n'y arrive pas, K propose "Qu'est-ce qu'on peut généraliser? Comparez avec le tableau"

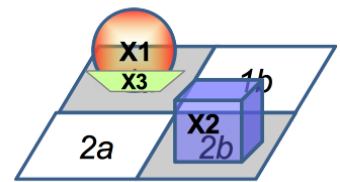
1.5 Debriefing:

- Qu'est-ce que vous pensez de **ce langage - les types, les propriétés et leur généralisation** ?
- Qu'est-ce que vous pensez de la représentation de l'action avec les préconditions et les effets?
- Est-ce que la représentation utilisée, vous semble correcte ?
- Est-elle claire ? Facile à comprendre ?
- Quelles améliorations apporteriez-vous ?

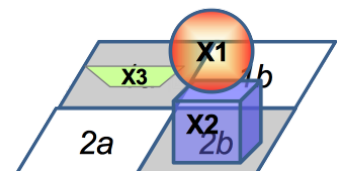
Niveau 1 – Tâche 2 - ajouter des contraintes nécessaire pour l'action

Nom d'objet	Type	Propriété (état initial)	Propriété généralisée
X3	tasse	est_sur(X3, 1a)	est_sur(tasse, position)
X1	balle	est_bleue (X1)	est_bleue(balle)
		est_sur(X1,X3)	est_sur(balle,tasse)
X2	cube	est_marron(X2)	est_marron(cube)
		est_sur(X2,2b)	est_sur(cube,position)
1a	position	non_vide(1a)	non_vide(position)
1b	position	est_vide(1b)	est_vide(position)
2a	position	est_vide(2a)	est_vide(position)
2b	position	non_vide(2b)	non_vide(position)

État initial



État final



Considérons maintenant un nouveau scénario.

K met les objets sur le damier.

2.1 Types/Propriétés:

- Observez ce nouvel état. Est-ce que le tableau est toujours complet/correct? Corrigez et complétez le tableau.
 - Si le tableau est faux ou incomplet, K demande "Est-ce que tu es sûr que ..."
 - Si le tableau est incomplet, K demande "Et la position du..." / "La propriété de...?"

K donne à S la fiche d'action II.

2.2 Actions:

Imaginons on veut déplacer la balle vers la position 2b. Modéliser cette action comme tout-à-l'heure.

3 Sous-tâches:

- Est-ce qu'on peut utiliser cette action de déplacement?
 - Si S dit oui, K lui demande "Montrez-moi l'action de déplacement" - **Tâche 2a**

Tâche 2a : est_empilable(balle,cube)	Tâche 2b : est_vide(position_arrivée)	Tâche 2c : est_empilable(tasse,cube)
<p>État final</p>	<p>État final</p>	<p>État final</p>

Tâche 2a : "Montrez-moi l'action de déplacement"

La balle ne va pas tenir sur le cube.

- Qu'est-ce qui ne va pas?
 - Si S ne trouve pas la bonne réponse, K dit "La balle ne tient pas sur le cube"

Donc, on ne peut pas utiliser l'action de déplacement dans ce cas.

K pose des questions qui guident S:

- Qu'est-ce qu'on peut ajouter pour qu'il déplace la balle seulement si elle tient sur le cube?
- Autrement dit, fais l'action seulement si la balle tient sur le cube. Donc, c'est une contrainte sur les propriétés des objets
- Il s'agit de quelle propriété entre la balle et le cube?
- Comment peut-on ajouter cette contrainte en tant que condition pour l'action?

Si S n'y arrive pas, K propose

- Peut-être on peut ajouter une précondition `est_empilable` de la même façon?
- En utilisant le langage avec le verbe/sujet, comment ça s'écrit?

Finalement,

- Quelle est cette condition généralisée?
 - `est_empilable(balle, cube)`

K explique "Donc, on a créé une action qui a deux condition pour déplacer la balle X1

1. la balle X1 est sur la position 1a
2. la balle est empilable sur le cube"

Transition à 2b/2c:

- On veut quand-même que la balle soit à la position 2b. Qu'est-ce qu'on peut faire?

Tâche 2b : "Il faut déplacer le cube"

- Quelle est la condition que l'on obtient si on déplace le cube?
 - S peut proposer "Autrement dit, la case est vide."
- Quelle est la condition que nous permet de déplacer la balle?
- Qu'est-ce qu'il faut ajouter?
- Quelle est sa formalisation généralisée?

K explique "Donc, on a créé une action qui a deux condition pour déplacer la balle X1

3. la balle X1 est sur la position 1a
4. la position d'arrivée 2b est vide"

! On peut enlever la condition `est_sur(X2, 2b)` qui dit que le cube X2 est sur 2b.

Transition à 2c:

- Qu'est-ce qu'on peut faire pour que la balle soit sur la position 2b sans déplacer le cube/ **au-dessus du cube?**

Tâche 2c : "Il faut déplacer la tasse avec la balle"

- Pourquoi peut-on déplacer la tasse sur le cube?
- Comment représenter dans notre langage que l'on peut mettre la tasse sur le cube, mais pas la balle?
- Quelle est la condition que l'on peut ajouter?

Finalement,

- Quelle est sa formalisation généralisée?
 - `est_empilable(tasse, cube)`

K explique "Donc, on a créé une action qui a deux condition pour déplacer la balle X1

5. la balle X1 est sur la tasse X3
6. la tasse X3 est empilable sur le cube"

Transition à 2b:

- Et si on pourrait pas empiler les objets, qu'est-ce qu'on peut faire pour que la balle soit à la position 2b?

Debriefing:

- Qu'est-ce que vous pensez de [ce langage - les types, les propriétés et leur généralisation](#) ?
- Qu'est-ce que vous pensez de la représentation de l'action avec les préconditions et les effets?
- Est-ce que la représentation utilisée, vous semble correcte ?
- Est-elle claire ? Facile à comprendre ?
- Quelles améliorations apporteriez-vous ?

Niveau 1 - Enchaînement des tâches

Considérons une nouvelle configuration. Vous avez un état initial.

- Définissez le but.
- Quelles sont les actions pour arriver de l'état initial au but? Montrez-les sur le damier.
- Voici, les actions qui sont disponibles. Quelles sont les actions dont on a besoin?
- Remplissez les actions et leurs préconditions et effets et mettez-les dans la bonne ordre.
- Montrez le lien entre les actions (leur précondition/effets)
 - Pourquoi avez-vous choisi les deux actions?

déplacer(cube,position,position)

déplacer(X1,1a,2b)

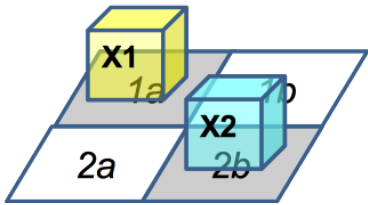
déplacer(X2,2b,1b)

déplacer(X2,2b,2a)

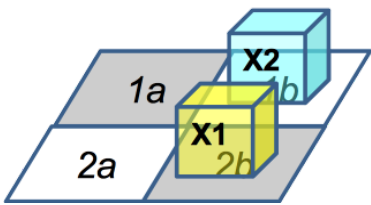
préconditions	action	effets
est_sur(___, ___)	déplacer(_X2____,	est_sur(____, ____)
est_vide(____)	____2b_, __1b____)	est_vide(____)

préconditions	action	effets
est_sur(___, ___)	déplacer(_X2____,	est_sur(____, ____)
est_vide(____)	____2b_, __1b____)	est_vide(____)

État initial



But



état initial		But
est_sur(X1, 1a)		est_sur(X1, 2b)
est_sur(X2,2b)		est_sur(X2,1b)
non_empilable(X1,X2)		
non_vide(1a)		est_vide(1a)
est_vide(1b)		non_vide(1b)
est_vide(2a)		est_vide(2a)
non_vide(2b)		non_vide(2b)

Niveau 2

K donne un tableau vide et repositionne les blocs sur le damier
Essayez de remplir le tableau à partir du nouveau scénario.

Interaction homme-machine

Bonjour, suite à l'expérimentation que vous venez de passer, je vais vous faire remplir un court questionnaire pour savoir ce que vous avez retenu de cette expérimentation. Merci de répondre le plus justement possible pour permettre une bonne analyse de vos données.

* Required

1. Nom *

2. Prénom *

3. Numéro d'étudiant *

4. Niveau d'étude *

5. Domaine de formation *

6. Numéro d'expérimentation *

7. Niveau et numéro de la dernière tâche
finie *

8. Quel est votre niveau de formation en programmation informatique? *

Mark only one oval.

☐

Votre domaine de formation est l'informatique (expert)

☐

Vous avez suivi des cours de programmation et votre domaine de formation n'est pas l'informatique (avancé)

☐

Vous savez utiliser des outils de bureautique et de communication (débutant)

☐

Vous n'avez aucune connaissance (formation ou pratique) en informatique

9. **Avez-vous déjà utilisé un langage de programmation? ***

Mark only one oval.

- ☐ Oui
- ☐ Non

10. **Si oui, lequel et à quelle occasion?**

11. **Avant l'expérimentation, pensiez-vous que vous aimeriez programmer un robot? ***

Mark only one oval.

- ☐ Pas du tout
- ☐ Un peu
- ☐ Plutôt
- ☐ Oui

12. **Etiez-vous à l'aise avec le vocabulaire utilisé durant l'expérimentation? ***

Mark only one oval.

- ☐ Pas du tout
- ☐ Un peu
- ☐ Plutôt
- ☐ Oui, beaucoup

13. **En utilisant le langage du robot, comment se représente-t-il la propriété "le cube X4 est bleu"? ***

14. En utilisant le langage du robot, comment se représente-t-il la propriété "le cylindre Y2 est sur la case 3b"? *

15. Comment peut-on généraliser cette propriété? *

16. Pouvez-vous expliquer la différence entre une propriété et une propriété généralisée? *

17. Pouvez-vous expliquer la différence entre la propriété `est_bleu(cube)` et `est_vide(case)`? *

Mark only one oval.

- ☐ Oui
☐ Non

18. Si oui, laquelle?

19. Pouvez-vous expliquer la différence entre la propriété `est_sur(cube,case)` et `est_empilable(cylindre, cube)`? *

Mark only one oval.

- ☐ Oui
☐ Non

20. Si oui, laquelle?

21. Est-ce qu'on peut avoir "est_vide(1b)" et "est_sur(X1,1b)" dans le même état? *

Mark only one oval.

☐ Oui

☐ Non

22. Expliquez pourquoi. *

23. Est ce qu'on peut avoir "non_empilable(X1,X2)" et "est_sur(X1,X2)" dans le même état? *

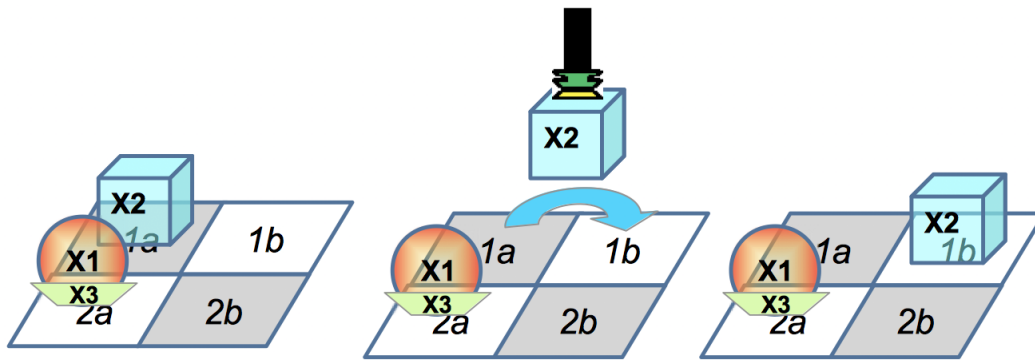
Mark only one oval.

☐ Oui

☐ Non

24. Expliquez pourquoi. *

Observez les trois graphiques suivantes qui représentent une action.



25. En utilisant le langage du robot, comment se représente-t-il l'action que vous voyez ci-dessus? *

26. Comment peut-on généraliser cette action? *

27. Pouvez-vous expliquer ce que signifie une précondition? *

28. En utilisant le langage du robot, pouvez-vous donner un exemple de précondition pour l'action de déplacement? *

29. Pouvez-vous expliquer la différence entre la précondition et l'effet d'une action? *

30. Observez les graphiques ci-dessus. Est-ce qu'on peut déplacer le cube X2 de la position 1a à 2a? *

Mark only one oval.

- ☐ Oui
- ☐ Non

31. Expliquez pourquoi. *

32. Quelle condition est nécessaire pour qu'on puisse déplacer le cube X2 de la position 1a à 2a? *

33. Seriez-vous capable de faire ce codage seul(e)? *

Mark only one oval.

- ☐ Pas du tout
- ☐ Un peu
- ☐ Plutôt
- ☐ Tout à fait

34. **Quel niveau de formation en programmation pensez-vous qu'il soit nécessaire d'avoir pour apprendre ce langage du robot? ***

Mark only one oval.

- ☐ Avoir une formation d'informaticien (expert)
- ☐ Avoir suivi des cours de programmation (avance)
- ☐ Pas de formation en programmation mais utiliser des outils de bureautique et de communication (debutant)
- ☐ Aucun

35. **Avez-vous rencontré des difficultés au cours de l'expérimentation? ***

Mark only one oval.

- ☐ Pas du tout
- ☐ Un peu
- ☐ Plutôt
- ☐ Oui, beaucoup

36. **Si oui, lesquelles?**

37. **Avez-vous des remarques ou des questions concernant cette expérimentation? ***

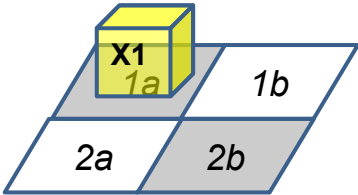
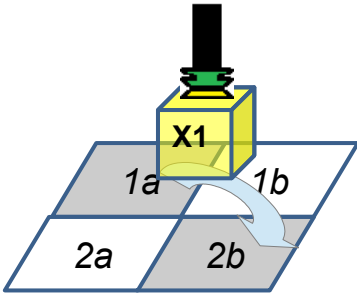
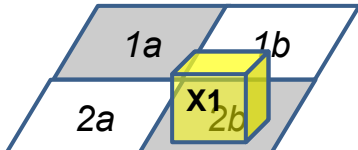
38. **Après l'expérimentation, pensez-vous que vous aimeriez programmer un robot? ***

Mark only one oval.

- ☐ Pas du tout
- ☐ Un peu
- ☐ Plutôt
- ☐ Oui

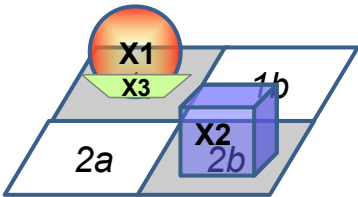
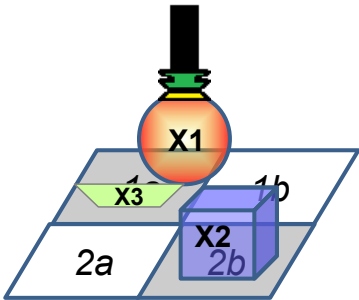
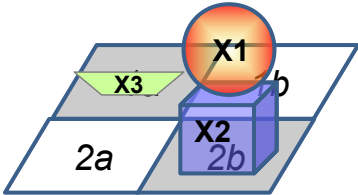
Action de déplacement d'un objet 1

généralisée

préconditions	action	effets
<ul style="list-style-type: none">est_sur(X1,1a)est_vide(2b)	déplacer(X1,1a,2b)	<ul style="list-style-type: none">est_sur(X1,2b)est_vide(1a)
préconditions	action généralisée	effets
<ul style="list-style-type: none">est_sur(cube,position)est_vide(position)	déplacer(cube, position_départ, position_arrivée)	<ul style="list-style-type: none">est_sur(cube,position)est_vide(position)
		

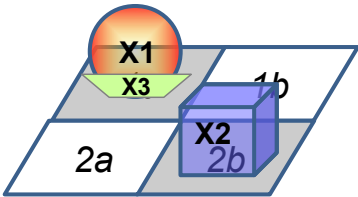
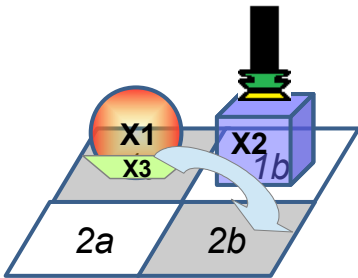
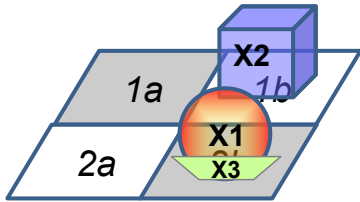
Action de déplacement d'un objet 2a

généralisée

préconditions	action	effets
<ul style="list-style-type: none"> est_sur(X3,1a) est_sur(X1,X3) est_sur(X2,2b) est_empilable(X1,X2) 	déplacer(X1,1a,2b)	<ul style="list-style-type: none"> est_sur(X3,1a) est_sur(X1,X3) est_sur(X2,2b) est_empilable(X1,X2)
préconditions	action généralisée	effets
<ul style="list-style-type: none"> est_sur(tasse,position) est_sur(balle,tasse) est_sur(cube,position) est_empilable(balle,cube) 	déplacer(balle, position_départ, position_arrivée)	<ul style="list-style-type: none"> est_sur(tasse,position) est_sur(balle,tasse) est_sur(cube,position) est_empilable(balle,cub e)
		

Action de déplacement d'un objet 2b

généralisée

préconditions	action	effets
<ul style="list-style-type: none"> est_sur(X3,1a) est_sur(X1,X3) est_sur(X2,2b) est_vide(2b) 	déplacer(X1,1a,2b)	<ul style="list-style-type: none"> est_sur(X3,1a) est_sur(X1,X3) est_sur(X2,2b) est_vide(1a)
préconditions	action généralisée	effets
<ul style="list-style-type: none"> est_sur(tasse,position) est_sur(balle,tasse) est_sur(cube,position) est_vide(position) 	déplacer(balle, position_départ, position_arrivée)	<ul style="list-style-type: none"> est_sur(tasse,position) est_sur(balle,tasse) est_sur(cube,position) est_vide(position)
		

Action de déplacement d'un objet 2c

généralisée

préconditions	action	effets
<ul style="list-style-type: none"> est_sur(X3,1a) est_sur(X1,X3) est_sur(X2,2b) est_empilable(X3,X2) 	déplacer(X1,1a,2b)	<ul style="list-style-type: none"> est_sur(X3,1a) est_sur(X1,X3) est_sur(X2,2b) est_empilable(X3,X2)
préconditions	action généralisée	effets
<ul style="list-style-type: none"> est_sur(tasse,position) est_sur(balle,tasse) est_sur(cube,position) est_empilable(tasse,cube) 	déplacer(balle, position_départ, position_arrivée)	<ul style="list-style-type: none"> est_sur(tasse,position) est_sur(balle,tasse) est_sur(cube,position) est_empilable(tasse,cube)

Appendix C

Resources for Pre-Experiment: Study 2 (Sec. 5.3)

The following documents were used in the second user study. As all experiments were conducted in French, all documents are in French:

- Experimental protocol
- Post-study questionnaire
- Action schema used

A summary of the resources used for this section can be found online:

- A demonstration of the Robot Programming process of our proposed framework: <https://youtu.be/DTm2YjiSNQM>
- The source code for the system implemented using the Wizard-of-Oz technique can be found online: https://github.com/ysl208/Baxter_PbD/

Protocole d'expérimentation pour la programmation du robot Baxter par démonstration

Personnes présentes

- X : expérimentateur
- Y : magicien d'Oz
- S : sujet de l'expérimentation

Accueil

Notez l'heure : _____

Bonjour S, je m'appelle X et je suis xx etc. Y va enregistrer l'expérience et mettre en route le robot Baxter.

Je travaille au sein de l'équipe MAGMA dont l'une des thématiques de recherche est la robotique. Mon projet de recherche consiste à étudier la collaboration Homme-Robot dans un environnement industriel. L'expérience que je vous propose aujourd'hui a pour objectif de voir si un opérateur humain sur une chaîne de production peut interagir avec le robot industriel Baxter pour lui apprendre à réaliser une tâche de déplacement d'un objet.

J'attire votre attention sur le fait que nous ne cherchons pas à évaluer votre travail ou vos performances. Votre participation va nous permettre d'évaluer et d'améliorer notre robot. Faites nous part de toutes vos remarques et de toutes vos difficultés : elles sont essentielles pour nous.

Dans un premier temps, je vous propose de vous présenter le robot Baxter. Dans un second temps, vous pourrez le manipuler pour vous familiariser avec lui. Ensuite, je vous demanderai de réaliser trois activités avec lui. Avant de nous séparer, je vous demanderai de remplir un questionnaire d'une dizaine de questions relatives à l'expérience.

Présentation de Baxter

Voici le robot Baxter, un robot utilisé dans les usines pour des tâches d'assemblage.

Voici une vidéo pour vous le montrer dans un contexte industriel :

<https://www.youtube.com/watch?v=oD9DE0HjMM4>

Quand on reçoit Baxter, il ne sait rien faire. Il faut tout lui apprendre. Pour l'instant, nous lui avons juste appris à chercher un objet sur la table à partir de sa couleur. Les activités que vous allez faire avec Baxter vont permettre de lui apprendre de nouvelles tâches.

Avez-vous des questions ?

Contexte expérimental

Nous allons maintenant vous montrer comment fonctionne Baxter puis vous le testerez.

Supposons que nous soyons sur une chaîne de production industrielle. Les objets arrivent de ce côté-là dans la zone de départ (D) qui est accessible pour l'opérateur pour réaliser une tâche de contrôle de qualité : mesure des cotes de l'objet.

Nous supposons également que les objets sont lourds ou dangereux et qu'il est souhaitable que l'opérateur humain ne manipule pas directement les objets. Le robot doit déplacer l'objet à la position d'arrivée (A) pour qu'un autre robot (que je simulerai) puisse le récupérer.

Il faut savoir que les objets ne sont pas empilables : deux objets ne peuvent être superposés l'un sur l'autre (risque d'endommagement des objets).

Démonstration et tests

Manipulation du bras par X

Maintenant, je vais vous montrer comment bouger le bras du robot. Sur la pince pneumatique du robot, il existe deux zones tactiles qui rendent déverrouiller le bras et qui permettent de bouger le bras très facilement.

Utilisez votre main gauche et placez le pouce sur un côté et les autres doigts sur l'autre. Puis, placez votre main droite sur le poignet du robot. Essayez de laisser la pince pneumatique dans la position verticale à environ 10 cm de la table. Vous pouvez déplacer le bras dans n'importe quelle position du moment que la pince reste dans la zone marquée par les lettres A et D. En appuyant sur le bouton blanc qui se trouve sur le côté du bras juste à proximité des zones tactiles, vous pouvez activer la pince pneumatique pour saisir un objet.

Manipulation du bras par S

Notez l'heure :

Je vous laisse essayer par vous-même. Approchez-vous du bras, appuyez sur le bouton blanc, déplacez l'objet à la position d'arrivée, relâchez l'objet et remonter le bras à 5 cm.

Vérifier le déplacement de l'objet selon les conditions de succès : on recommence en cas d'échec :

- **Succès** : S a bien guidé le bras et effectué un déplacement continu,
- **Echec** : proposer à S de réessayer la manipulation. Si S montre des difficultés, refaire plusieurs fois et éventuellement reprendre l'explication et remontrer.

DEBRIEFING : noter des réponses de S

- Est-ce que vous avez des questions ?

- Est-ce que vous vous sentez à l'aise avec Baxter ?
- Est-ce qu'il vous semble difficile de manipuler le bras de Baxter ?
- Quelles améliorations apporteriez-vous ?

Expérimentation

Maintenant, nous allons vous demander d'apprendre à Baxter à réaliser une nouvelle tâche : déplacer un objet. Tout ce qu'il sait faire jusqu'à présent, c'est chercher un objet placé en (D) ou (A).

X replace l'objet dans la zone de départ et le bras au-dessus.

Rappelez-vous, vous êtes sur une chaîne de production industrielle. Les objets arrivent de ce côté noté (D) pour départ et nous souhaitons apprendre à Baxter comment déplacer l'objet de la zone (D) à la position d'arrivée noté (A) pour que robot suivant sur la chaîne de production (joué par moi) puisse le récupérer. Je vous rappelle que les objets ne peuvent pas être empilés.

Enregistrement du déplacement de l'objet avec Baxter

Notez l'heure : _____

Je vous propose maintenant que vous êtes à l'aise avec Baxter d'enregistrer le déplacement de l'objet de la position de départ à la position d'arrivée comme nous venons de le faire. Ne vous faites pas de souci, en cas de problème, nous pourrions le refaire.

Etes-vous prêt de commencer l'enregistrement ?

Y démarre enregistrement sur l'ordinateur

Voilà Baxter hoche la tête quand il est prêt : vous pouvez commencer l'enregistrement.

Le sujet montre le déplacement de l'objet de la position de départ (D) à la position d'arrivée (A)

Quand l'opérateur a relâché l'objet, terminer l'enregistrement.

Vérifier le déplacement selon les conditions de succès et répéter en cas d'échec :

- **Succès** : il a bien guidé le bras et effectué un déplacement continu,
- **Echec** : on peut enregistrer le même déplacement une autre fois. Remplacez l'objet à la position de départ.

Voilà Baxter hoche la tête quand l'enregistrement est fini.

DEBRIEFING : noter les réponses de S

- Comment s'est passé l'enregistrement ?
- Quelles améliorations apporteriez-vous ?

On va rejouer le déplacement pour voir si Baxter l'a bien appris. Remplacez l'objet à la position de départ.

Replay : le robot déplace l'objet à la position d'arrivée (A)

- Est-ce que c'est bien le déplacement que vous lui avez appris ?
- Que pensez-vous de cet apprentissage ?
- Quelles améliorations apporteriez-vous ?

Compréhension par S de la sémantique des tâches apprises à Baxter

Notez l'heure : _____

Voilà comment Baxter se représente ce que vous venez de lui apprendre.

X montre une fiche avec les préconditions de la tâche.

Il a compris que l'objet rouge doit être à la position de départ (D) pour effectuer le déplacement.

Après le déplacement, il a compris que l'objet rouge n'est plus à la position de départ (D) mais à la position d'arrivée (A).

Schéma de la tâche apprise :

- Préconditions :
 - l'objet rouge est à la position de départ (D),
- Effets :
 - l'objet rouge n'est plus à la position de départ (D),
 - l'objet rouge est à la position d'arrivée (A).

En plus du texte, montrer la représentation graphique.

DEBRIEFING : noter des réponses de S

- Pensez-vous que le robot a bien compris ce que vous vouliez lui apprendre ?
- Est-ce que la représentation utilisée, vous semble correcte ?
- Est-elle claire ? Facile à comprendre ?
- Quelles améliorations apporteriez-vous ?

Compréhension par S des modifications à apporter aux préconditions pour généraliser la tâche apprise à des objets de différentes couleurs

Notez l'heure : ____

Enlevez l'objet rouge de la table. Posez l'objet bleu à la position de départ (D)

Je vous propose de demander à Baxter de déplacer maintenant un objet bleu.

- Que va faire le robot selon vous ?

Si S ne trouve pas la solution, essayez de faire exécuter le déplacement par Baxter

Nous allons demander à Baxter de déplacer l'objet bleu.

Démarrer le déplacement du bras : « move arm to colour »

- Pourquoi a-t-il refusé de déplacer l'objet ?

Remontrer le schéma de tâche apprise

Rappelez-vous ce que Baxter avait compris de votre démonstration.

- Que faudrait-il dire à Baxter pour qu'il puisse déplacer un objet bleu ?
- Comment représenter la tâche pour déplacer un objet quelle que soit sa couleur ?

Si le sujet ne trouve pas la solution, proposez la modification de la condition.

Voilà ce que le robot a compris de ce vous avez dit : il a compris que la couleur n'a aucune importance pour effectuer le déplacement.

X montre la 2e schéma :

- Préconditions :
 - l'objet est à la position de départ (D),
- Effets :
 - l'objet n'est plus à la position de départ (D),
 - l'objet est à la position d'arrivée (A).

DEBRIEFING : noter des réponses de S

- Pensez-vous que c'est suffisant pour que Baxter puisse déplacer l'objet bleu ?
- Que va faire le robot maintenant ?

Voyons ce que Baxter va faire maintenant.

Y démarre Baxter pour déplacer l'objet bleu.

- Qu'en pensez vous ?

- Est-ce que ça vous paraît normal ?
- Quelles améliorations apporteriez-vous ?

Compréhension par S d'une contrainte nécessaire au déplacement d'un objet

Notez l'heure : _____

Placez l'objet rouge à la position de départ (D). L'objet bleu se trouve toujours à la position d'arrivée (A).

Je vous propose de demander à Baxter de déplacer l'objet rouge.

- Que va faire le robot selon vous ?

Si S ne trouve pas la solution, rejouer le déplacement : démarrer le robot qui déplace l'objet rouge au dessus de l'objet bleu.

Rappelez à S qu'une des contraintes de la chaîne de production est que les objets ne sont pas empilables.

Remonter le schéma de la tâche apprise.

- Pourquoi Baxter a-t-il empilé 2 objets ?
- Qu'est-ce qu'il faut faire pour déplacer l'objet bleu ?

Si S ne trouve pas la solution, proposez la modification de la précondition.

- Peut-on représenter la tâche autrement pour que Baxter ne viole pas la contrainte d'empilement ?

Voilà ce que Baxter se représente ce que vous avez dit.

X montre le nouveau schéma :

- Préconditions :
 - l'objet est à la position de départ (D),
 - **la position d'arrivée (A) est vide,**
- Effets :
 - l'objet n'est plus à la position de départ (D),
 - l'objet est à la position d'arrivée (A).

DEBRIEFING : noter des réponses du sujet

- Pensez-vous que c'est suffisant pour que Baxter puisse déplacer un objet ?
- Que va faire Baxter maintenant ?

Baxter émet un message pour dire qu'il ne peut pas effectuer le déplacement.

- Qu'en pensez-vous ?
- Est-ce que ça vous paraît normal ?
- Quelles améliorations apporteriez-vous ?

Compréhension par S de ce que peut dorénavant faire Baxter avec la nouvelle tâche apprise

Notez l'heure : _____

X déplace les barrières blanches et les deux objets.

Rappel du contexte : maintenant, Baxter sait faire deux choses : chercher un objet (c'est nous qui lui avons appris avant votre arrivée), et déplacer des objets grâce à vous.

Je vous propose de demander à Baxter de permuter l'objet bleu avec l'objet bleu de la position (D) à la position (A).

Nous sommes toujours sur une chaîne de production et c'est le même Baxter à qui vous avez appris le déplacement. Les objets arrivent les uns après les autres mais, parfois, il y a des problèmes de rangement : il faut que les objets rouges soient devant les objets bleus. Baxter doit alors réordonner des objets.

X montre le déplacement des deux objets sur la table

- Pensez-vous que Baxter va réussir ?
- Que va-t-il faire ?
- Si c'était vous qui deviez permuter les objets avec un seul bras, que feriez-vous ?

Y démarre la permutation des objets.

- Pensez-vous que Baxter a agi de manière intelligente ?
- Quel a été son raisonnement ? Quelles tâches a-t-il utilisé ?

Je vous remercie. Nous avons fini avec l'expérience. Installez-vous pour répondre à un questionnaire.

Donner le questionnaire à S.

Questionnaire sur la programmation d'un robot Baxter par démonstration

Nom :

Prénom :

Quel est votre niveau d'études ?

- Bac + 5
- Bac + 2
- Bac ou diplôme professionnel

Quel est votre domaine de formation ?

- Arts, lettres, langues
- Droit, économie, gestion
- Sciences humaines et sociales
- Sciences, technologies, santé

Quel est votre niveau de formation en programmation informatique ?

- Votre domaine de formation est l'informatique (expert)
- Vous avez suivi des cours de programmation et votre domaine de formation n'est pas l'informatique (avancé)
- Vous savez utiliser des outils de bureautique et de communication (débutant)
- Vous n'avez aucune connaissance (formation ou pratique) en informatique

Avez-vous déjà utilisé un robot capable de saisir par lui-même un objet ?

- Oui
- Non

Si oui, quel était le type de ce robot ?

- Jouet ou robot à construire soi-même
- Robot compagnon (par exemple Nao)
- Robot industriel ou de recherche

Avez-vous déjà entendu parlé de planification automatique ?

- Oui
- Non

Si oui, à quelle occasion ?

Le robot Baxter vous semble-t-il facile à manipuler physiquement ?

- Pas du tout facile
- Plutôt pas facile
- Plutôt facile
- Tout à fait facile

Est-ce que le robot Baxter vous semble adapté à la manipulation par des opérateurs sur une chaîne de production ?

- Pas du tout adapté
- Plutôt pas adapté
- Plutôt adapté
- Tout à fait adapté

Pensez-vous que le robot Baxter agisse de manière intelligente ?

- Pas du tout
- Un peu
- Plutôt
- Oui, beaucoup

Pensez-vous avoir appris au robot Baxter à réaliser de nouvelles tâches ?

- Oui
- Non

Si oui, lesquelles ?

Pouvez-vous expliquer comment Baxter se représente la nouvelle tâche apprise ?

- Pas du tout
- Un peu
- Plutôt
- Oui, beaucoup

Pouvez-vous expliquer comment Baxter apprend une nouvelle tâche à partir de plusieurs exemples ?

- Pas du tout
- Un peu

- Plutôt
- Oui, beaucoup

Pouvez-vous expliquer comment Baxter se représente la pré-condition d'une tâche ?

- Pas du tout
- Un peu
- Plutôt
- Oui, beaucoup

Quel niveau de formation en programmation pensez-vous qu'il soit nécessaire d'avoir pour apprendre une nouvelle tâche au robot Baxter ?

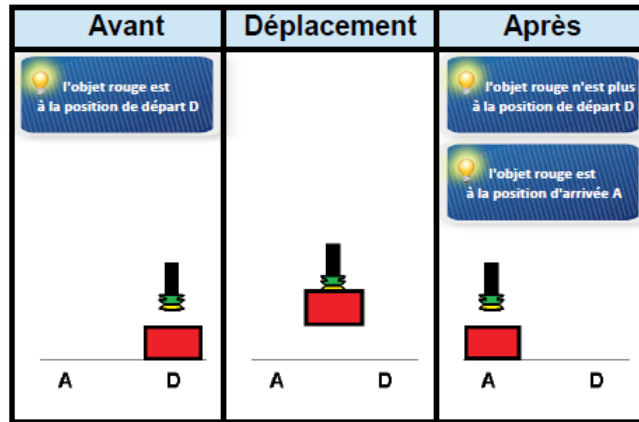
- Avoir une formation d'informaticien (expert)
- Avoir suivi des cours de programmation (avancé)
- Pas de formation en programmation mais utiliser des outils de bureautique et de communication (débutant)
- Aucun

Avez-vous rencontré des difficultés au cours de l'expérimentation ?

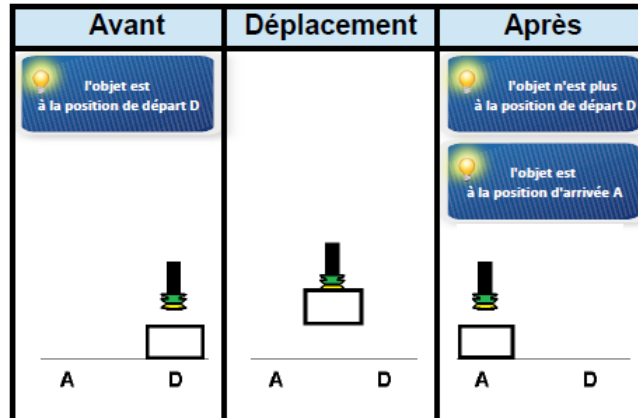
- Pas du tout
- Un peu
- Plutôt
- Oui, beaucoup

Si oui, lesquelles ?

Action de déplacement d'un objet I



Action de déplacement d'un objet II



Action de déplacement d'un objet III

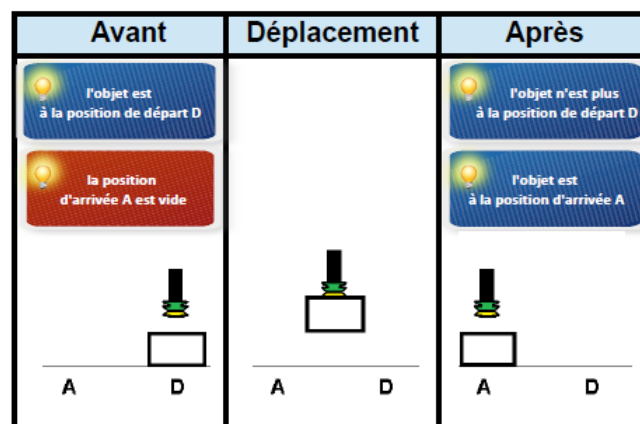


Figure C.1: Action model used in experiments

Appendix D

Resources for User Evaluation (Chapter 6)

The resources used for evaluating the teaching strategies presented in Chapter 6 are as follows:

- Python code for simulating the teaching strategies: <https://github.com/ysl208/organisingtasks/>
- The graphical interface used for the AMT user study can be found on Codepen: <https://codepen.io/ysliang208/project/editor/DYryzp#0>

Appendix E

Resources for User Evaluation (Sec. 7.5)

The following documents were used in the final user study (all documents are in English):

- THEDRE brainstorming guide
- THEDRE experimental protocol guide
- Slides used during the experiment for introduction and tasks
- Pre-study questionnaire
- Post-study questionnaire: including questions from the SUS
- The source code for the iRoPro system implementation can be found online: <https://github.com/ysl208/iRoPro/tree/cond>

Guide de « Brainstorming » (bloc 1 et bloc 2)

Questions pour vous guider	Vos Réponses
Quel est le problème ?	It is extremely difficult to program robots for specific end-user applications ranging from manufacturing environments to personal homes. There are many companies that are robot resistant as they do not have the trained personnel to fully exploit the robots, and non-expert users cannot program the robots. The problem that we address is how can a non-expert user program a robot.
Dans quels contextes se pose le problème ? quand ? où ?	This problem arises in many small- to medium industrial/manufacturing environment when deploying a robot, but also in general settings where a robot has to be programmed by a non-expert user.
D'où émane la demande ?	In general it comes from end-users without robot programming knowledge who want to customise their robot for specific tasks, e.g. manufacturing industries, supermarket stores, etc.
Qui est concerné par le problème ?	end-users without robot programming knowledge who want to customise their robot for specific tasks
Comment pouvez vous résoudre le problème ?	create an intuitive robot programming framework for non-experts that allows teaching it new actions by kinesthetic demonstration and interaction with a graphical interface, and the use of automated planning
Pourquoi est-il important de résoudre ce problème au niveau académique ?	proof of concept to show that this is a feasible robot programming approach for non-experts
Quel est l'intérêt de répondre à ce problème par rapport aux attentes de la société ?	companies hesitate to use robots because of the lack of programming experts, allowing non-experts to program robots would increase productivity, allow untrained personnel to work with robots and therefore open a market to program robots without needing to write code
Quels sont les auteurs ou références bibliographiques à utiliser ?	[1] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Robot programming by demonstration. In Springer Handbook of Robotics, pages 1371–1394. Springer, 2008 [2] Argall, B. D., Chernova, S., Veloso, M., & Browning, B. (2009). A survey of robot learning from demonstration. Robotics and autonomous systems, 57(5), 469-483. [3] Abdo, N., Kretschmar, H., Spinello, L. and Stachniss, C. Learning manipulation actions from a few demonstrations. In 2013 IEEE International Conference on Robotics and Automation (ICRA), (pp. 1268-1275). IEEE. [4] M. Ghallab, D. Nau, and P. Traverso. Automated planning: theory & practice. Elsevier, 2004 [5] Alexandrova, S., Cakmak, M., Hsiao, K., & Takayama, L. (2014, July). Robot Programming by Demonstration with Interactive Action Visualizations. In Robotics: science and systems. [6] 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.
Qu'est ce qui a été fait dans le domaine académique pour résoudre le problème ?	Programming by demonstration [1,2] has been commonly applied to allow end-users to teach robots task goals or policies by demonstration. Recent work has focused on industrial manipulators [6] and mobile manipulators [5]. Most closely related to our work are Alexandrova et al. [5], who created an end-user programming framework with an interactive action visualisation but without task planning; and Abdo et al. [3] that teach manipulation actions from few demonstrations and a symbolic planner is used to achieve goals, but does not provide an graphical interface for end-users to set their own goals.
Quelles sont les méthodologies présentes dans les publications ? Comment la construction et l'évaluation ont été réalisées ?	Abdo et al. [3] takes multiple demonstrations and uses k-means and entropy to deduce action conditions from demonstrations. They evaluated their system with experiments on teaching the robot to stack blocks, pour from a bottle and to open a door programmed. Alexandrova et al. [5] interactive action visualization to program the robot and evaluates the system on 12 benchmark tasks as well as a user study (N=10) for box closing tasks, stacking cups, and putting objects into a box. Stenmark et al. [6] uses assembly tasks to pick and stack lego blocks and compares the performance of non-experts with reusable tasks (N=3x7 participants).
Quelles sont les avancées technologiques sur le sujet ?	dynamic activable tool in the form of a software prototype
Qu'est ce qui a été fait dans le domaine technique pour résoudre le problème ?	The main motivation for my thesis is to allow end-users to program robots, without writing explicit code. We implemented a system for teaching robots atomic actions with conditions that can be used by symbolic planners to solve more complex problems. Using the system, the user interacts with the graphical interface to customise the taught action and its conditions, to activate the robot's perception to detect objects on the table, to teach it a new action by kinesthetic manipulation of its arms, and to activate the robot's action condition generation. The robot's learning algorithm generalises the actions to different environments and reuses them to autonomously generate solutions to problems that go beyond the learned actions. The user can create a new problem with a goal to achieve (e.g. stacking objects) and execute the actions on the robot.

<p>Par rapport au problème posé, quels sont les manques ? Que reste-t-il à résoudre ?</p>	<p>The state of the art in Programming by Demonstration uses demonstrations to teach robots whole action sequences to reach a goal. When the goal changes, a new action sequence needs to be taught. This can be very time consuming.</p> <p>This is why we want to teach the robot atomic actions and their conditions, which together can be combined with a task planner to achieve a variety of different goals.</p> <p>Now that we have created a system that addresses this problem, we need to evaluate its usability in terms of end-user experiments as a proof of concept.</p>
<p>A quoi ces résultats vont-ils servir ? et à qui ?</p>	<p>The results from the user study in the form of human-robot experiments will contribute to the evaluation of this proposed framework and the proof of concept.</p>
<p>Quelle valeur ajoutée allez vous apporter ?</p>	<p>Rédiger ce que votre recherche ajoutera à la connaissance scientifique actuelle.</p> <p>The implemented real-world system to be used for robot programming by non-experts together with its user study will contribute to the robotics research community to compare with state-of-the-art end-user robot programming frameworks and show that it is possible for end-users without programming knowledge to teach robots new actions for complex tasks.</p>

Protocole Expérimental		
Catégorie	Éléments à renseigner	Description de l'élément
Suivi du document	Date de création : 04/11/2018	Date à laquelle le document est créé
	Dates de modification	Dates des modifications successives du documents
	auteur(s) du document Ying Siu Liang - Experimenter	Acteurs internes : Nom et rôle
Objectifs	Nom de l'expérimentation: End-user Robot Programming of Action models for Symbolic planning tasks	Donner un nom à l'expérimentation
	Objectif de l'expérimentation: The results from the user study in the form of human-robot experiments will contribute to the evaluation of the implemented robot programming framework and the proof of concept.	Décrire à quoi cette expérimentation va servir
	Questions ou hypothèses : H1 The user can teach the robot a new action The user can teach the robot an action by demonstration The user successfully executed the taught action on the robot at least once The user can associate correct preconditions & effects to the robotH2 The user can create a new problem on the GUI The user can define the problem states The user can define the goal states to solve this problem The user understands the plan generated by the automated plannerH3 The user understands the system If no plan is generated, the user can troubleshoot on their own The user knows can navigate within the system on their own	Indiquer les questions et les hypothèses qui devront trouver des éléments de réponses lors de cette expérimentation
Outils et composants activables	The dynamic activable tool is the robot programming software prototype) and will evaluate the following components: Action: perception demonstration action model creation Problem: initial state goal state generated plan	Lister les différents composants de l'outil activables qui vont être construits ou évaluer lors de l'expérimentation.
	Action: perception: activate object detection on the robot and verify objects are correct demonstration: move robot's arm to teach the action and save arm poses action model: generate conditions and verify/modify them Problem: initial state: activate object detection on the robot and verify objects are correct goal state: enter predicates to describe the goal state generated plan: verify the plan is correct and execute on robot	Indiquer l'état des composants et comment l'utilisateur peut les utiliser lors de l'expérimentation (p.ex., statique, dynamique, non manipulable)
Production des données	Méthodes de production : quantitative user tests + questionnaire	Indiquer le type de méthode choisie (qualitatives, quantitative ou mixtes). Préciser les méthodes de production utilisées (p.ex., questionnaire, tests utilisateurs, construction de maquettes)
	Matériel technique : video+audio recording of all experiments	Indiquer le matériel technique nécessaire à avoir pour la capture des données (p.ex., caméra, enregistreur)
	Matériel expérimental protocol, questionnaire, consent form	Lister le matériel expérimental à construire pour réaliser l'expérimentation (p.ex., présentation, questionnaire)
	Matériel et données produites video recording, number of tasks successfully implemented, time taken for each task	Indiquer tout le matériel et les données produits lors de cette passation (schéma, audio, traces)
Utilisateurs	Nombre d'utilisateurs : 15-20	Indiquer le nombre d'utilisateurs prévus pour cette passation.
	Profil des utilisateurs : English speakers of any background We will use a logic test to evaluate their reasoning abilities as a benchmark	Indiquer qui sont les utilisateurs qui vont être mobilisés et pourquoi
	Lieu de passation lab	Indiquer le lieu où la passation aura lieu (p.ex., in lab, in situ)

	Recrutement users will be recruited from LIG, other labs, and acquaintances (lab's family members, Grenoble PhD communities, sport groups etc.) by sending out emails or posting flyers	Indiquer comment le recrutement des utilisateurs est fait
	Mode passation experiments will be one by one	Indiquer si les utilisateurs sont consultés seuls ou en groupe
	Ethique et déontologie we will have each user sign a consent form at the start of the experiment	Indiquer les démarches auprès de la CNIL ou d'un comité d'éthique pour déclarer l'expérimentation
Planning	Planning 06/11 Send Nadine documents 12/11 Write experimental protocole run pre-tests with Damien/Humbert/Sylvie 20/11 final meeting with Nadine, recruit participants 21/11 prétests with team members 26/11 run experiments 03/12 evaluate results	Indiquer à gros grain les étapes de l'expérimentation. (un planning précis des jours et dates des passations est aussi établi).
Analyse des données	Outils de codage excel spreadsheet (for noting down start/end times, tasks successfully completed etc.) questionnaire on google docs	Lister les outils nécessaires pour coder les données (p.ex., grille d'annotation)
	Méthodes et outils d'analyse de données googcsle do provides data analysis for the questionnaire excel spreadsheets to generate charts entered	Lister les outils et méthodes d'analyse pressentis pour analyser les données. Préciser le plan de traitement prévisionnel

Welcome to the Robot Programming User Study

Ying Siu LIANG, 3rd year PhD, HAWAII

Robot Programming User Study

- We are **not** trying to evaluate your performance
- Evaluate and improve **our** robot programming system
- Comments & difficulties you're facing during the experiment are important!

Robot Programming User Study - Ying Siu LIANG, PhD, HAWAII

Overview

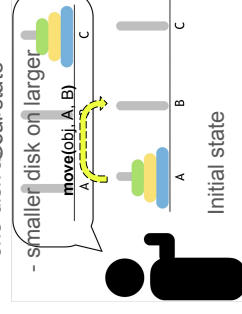
1. Introduction to Robot Programming
2. How to program Baxter
3. User tasks
4. Post-study questionnaire

Robot Programming User Study - Ying Siu LIANG, PhD, HAWAII

How do we teach humans?

2. Problems with robot demonstrations:

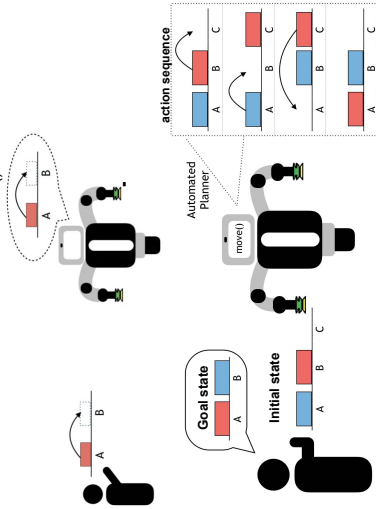
- one disk at a time



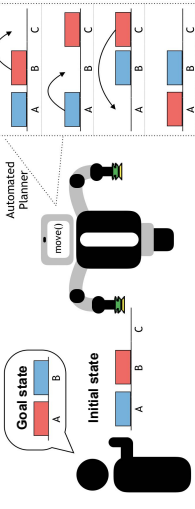
How do we teach robots?

1. Actions

move() + conditions!

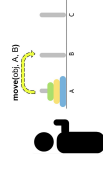


2. Problem



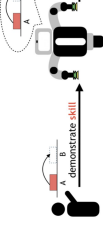
Humans

1. Learn Actions + Conditions



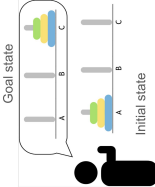
Robots

move() + conditions!



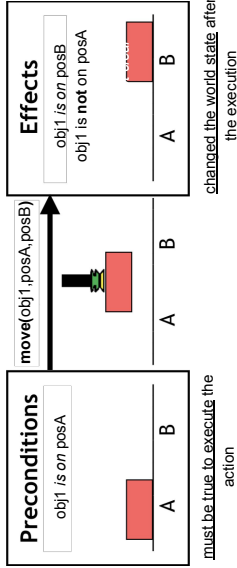
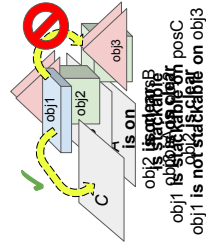
2. Solve Problems

+ Initial state + Goal state



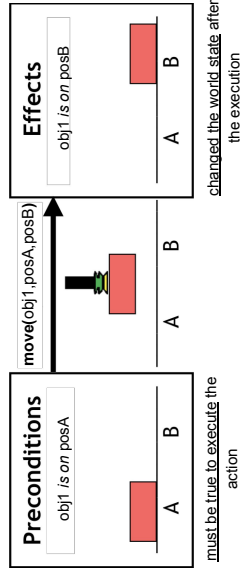
Predicates

Predicates

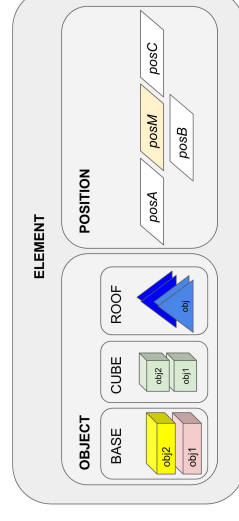
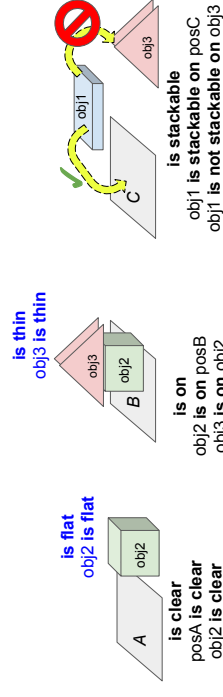
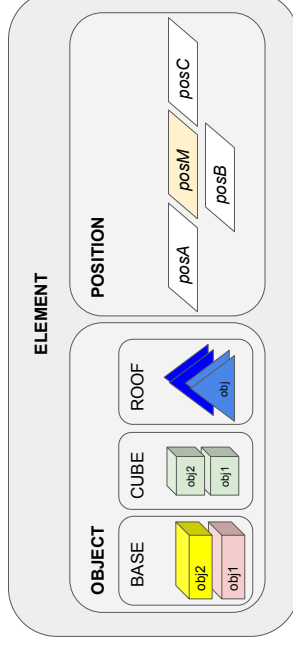


action(ELEMENT, ELEMENT, ...)

```
move(CUBE, POSITION, POSITION)
```



```
move(BE8ECP,OSDISK,NO POSITION,NO N)
```



Questions?

<https://goo.gl/forms/or7WoGqZeMl1KVT92>

Experimental Context

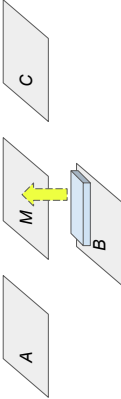
- Production line (positions A,B,C,M)
- Object types
 - Roof
 - Cube
 - Base
- Baxter does not know how to pick up objects
- Baxter grippers
 - Electric
 - Vacuum
- Press ☐ start ☐ stop

Manipulating Baxter

<https://youtu.be/oD9DE0H1MM4?t=28>

Move a BASE object

Program Baxter to move the BASE from position B to position M

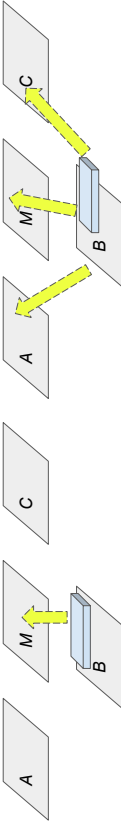


1. Create **Action**
2. **Perception step**
 - a. Click on DETECT
 - b. Verify all detected objects and their types
3. **Demonstration step**
 - a. REPLAY action at least once
4. **Conditions step**
 - a. Detect effects
 - b. Save conditions
5. Rename action "move-base" & Save

Note start time: _____

Move a BASE object

Program Baxter to move the BASE from position B to position M



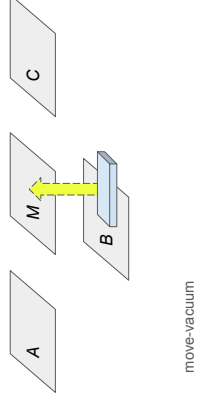
1. Create **Action**
 2. **Perception step**
 - a. Click on DETECT
 - b. Verify all detected objects and their types
 3. **Demonstration step**
 - a. REPLAY action at least once
 4. **Conditions step**
 - a. Detect effects
 - b. Save conditions
 5. Rename action "move-base" & Save
-
1. Create **Problem**
 2. **Initial states step**
 - a. Click on DETECT
 - b. Verify object types and initial states
 3. **Goal states step**
 - a. Add goal states
 4. **Generated Plan step**
 - a. Verify action sequence
 - b. EXECUTE plan
 5. Rename problem "rearrange" & Save

Note start time: _____

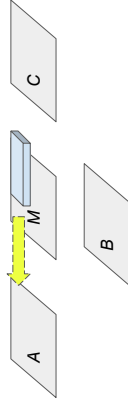
What if we want to move to any position?

Experiment tasks

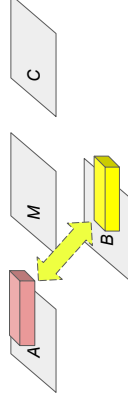
Teach Baxter an Action to move a BASE object



Move BASE object to any position (e.g. A)



Generate a plan to swap positions of two BASE objects

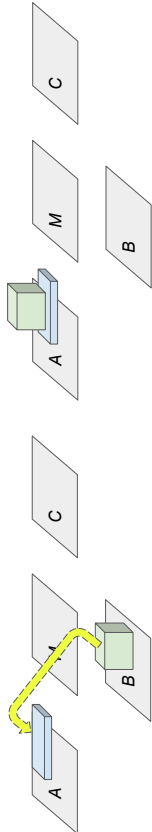


Define the goal states and let Baxter figure out the action steps

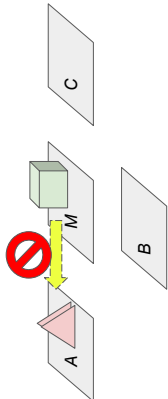
Hint: How can Baxter come up with the right steps?

(NO EXECUTION)

Stack CUBE object on BASE

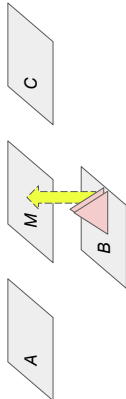


Modify the existing move action



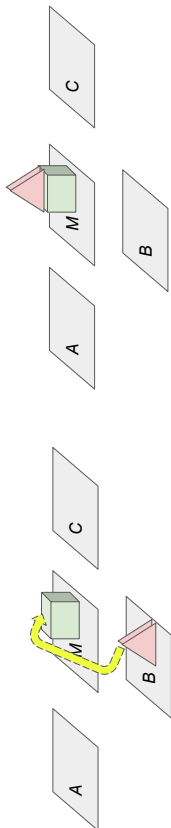
Modify the existing move action so that Baxter would not stack the CUBE object if it is a ROOF (NO EXECUTION)

Move ROOF object to position M

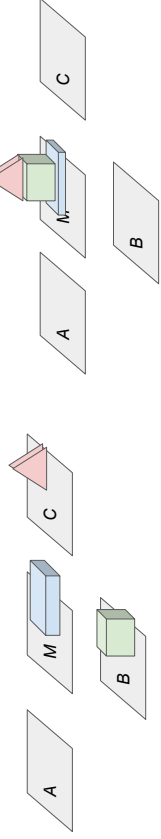


move-grip

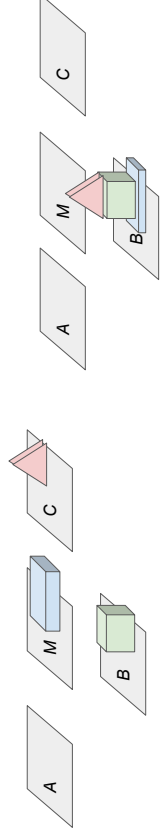
Stack ROOF object on CUBE



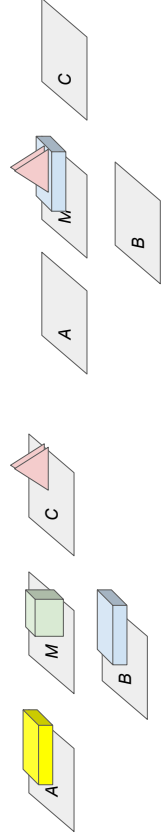
Final task: Build a house from all objects



Final task: Build a house from all objects - part 2



Bonus task: Have BASE and ROOF on position M



Robot Programming Study Questionnaire

* Required

1. Participant number *

2. Age *

3. Gender *

Mark only one oval.

- ☐ Female
- ☐ Male
- ☐ Prefer not to say
- ☐ Other: _____

4. Main field of study *

Mark only one oval.

- ☐ Business, Management, and Law
- ☐ Earth and Environmental Sciences
- ☐ Engineering
- ☐ Geography, Urban and Regional Planning
- ☐ Health Sciences, Life Sciences, and Chemistry
- ☐ Mathematics and Computer Science
- ☐ Physics and Materials Science
- ☐ Other: _____

5. What is your level of familiarity with programming languages? *

Mark only one oval.

- ☐ 1 - No experience
- ☐ 2 - Novice: you have experience with MS Word, Excel, PowerPoint, etc.
- ☐ 3 - Intermediate: you have taken a programming course before (<2 year experience)
- ☐ 4 - Advanced: you are currently pursuing a degree in Computer Science (>=2 years experience)
- ☐ 5 - Expert: you have completed a degree in Computer Science

6. What is your level of familiarity with symbolic planning languages (e.g. STRIPS, PDDL)? *

Mark only one oval.

- ☐ 1 - No experience
- ☐ 2 - Novice: you have heard of them before but never used them.
- ☐ 3 - Intermediate: you have taken a course before
- ☐ 4 - Advanced: you have worked on a project before (<2 years experience)
- ☐ 5 - Expert: you are actively working on a project (>=2 years experience)

7. Have you previously programmed a robot? *

Mark only one oval.

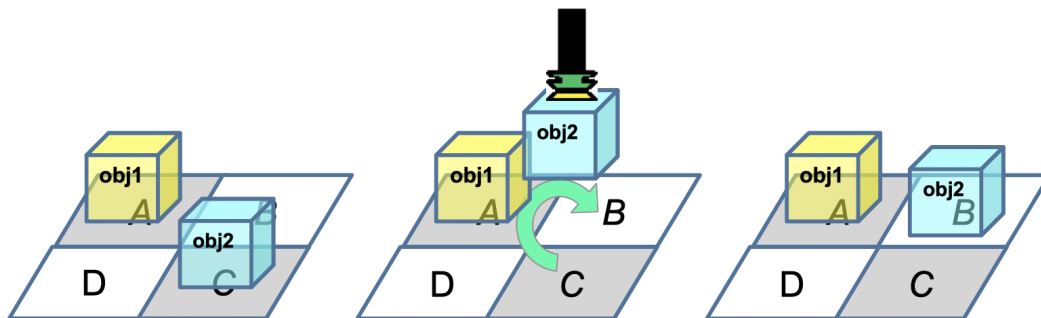
- ☐ Yes
- ☐ No

8. If yes, what kind of robot and which platform/programming language did you use?

Concepts and terminology

The following questions are to make sure you understood the concepts needed for the next tasks. Try to answer what seems correct.

Observe the image sequence below describing an action to move an object.

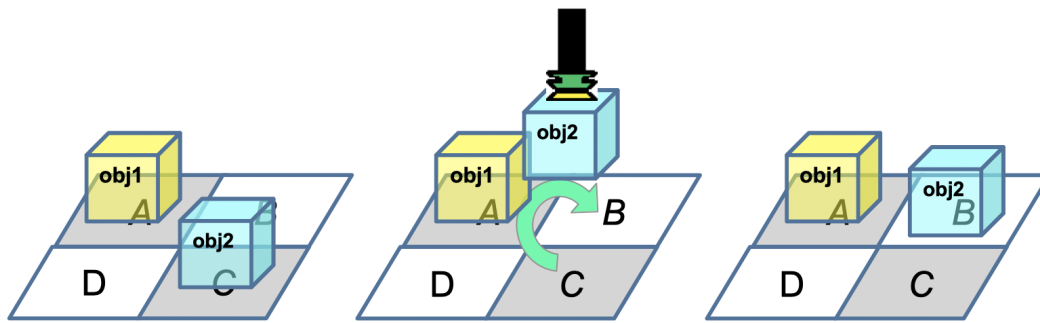


9. Which action corresponds to the above described move action?

Mark only one oval.

- ☐ move(obj1, posC, posB)
- ☐ move(obj2, posC, posB)
- ☐ move(obj1, obj2, posC, posB)
- ☐ move(obj1, obj2, posC, posB, posA, posD)

(same action as above)



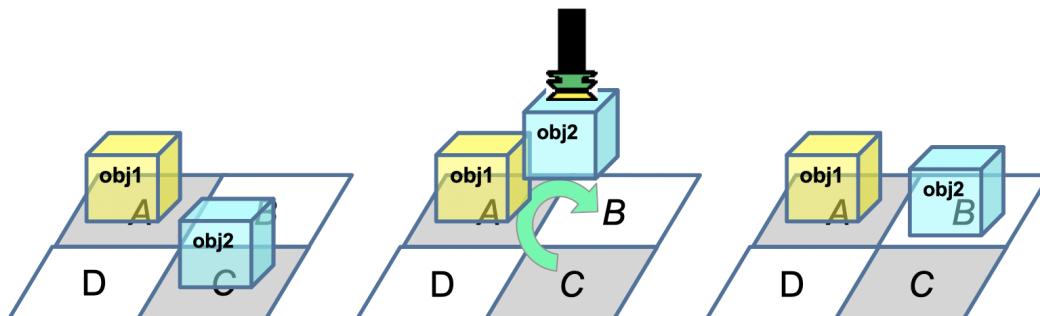
10. Tick all predicates that are required as preconditions for the given move action.

Hint: tick only those that are necessary for the given action *

Check all that apply.

- ☐ obj1 is clear
- ☐ obj2 is clear
- ☐ obj1 is on posA
- ☐ obj2 is on posC
- ☐ posA is not clear
- ☐ posB is clear
- ☐ posC is not clear
- ☐ obj1 is stackable on posA
- ☐ obj1 is stackable on posB
- ☐ obj2 is stackable on posB
- ☐ obj2 is stackable on posC

(same action as above)



Robot Programming - Post-Study Questionnaire

Please submit feedback regarding the study you have just completed, including feedback on the robot programming process, user interface, or any difficulties you encountered.

* Required

1. Participant number *

2. Overall system usability *

Mark only one oval per row.

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
I think that I would like to use this system frequently.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I found the system unnecessarily complex.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I thought the system was easy to use.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I think that I would need the support of a technical person to be able to use this system.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I found the various functions in this system were well integrated.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I thought there was too much inconsistency in this system.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I would imagine that most people would learn to use this system very quickly.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I found the system very cumbersome to use.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt very confident using the system.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I needed to learn a lot of things before I could get going with this system.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

3. Overall robot programming experience *

Mark only one oval per row.

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
It was easy to manipulate the Baxter's arms	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The robot programming process is well-adapted for workers on the assembly line	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Baxter's behaviour was intelligent	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I believe that I have taught Baxter a new task	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I can explain how Baxter represented the new action	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I can explain how Baxter learned a new action from my demonstration	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I can explain how Baxter represented the preconditions and effects of the new action	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I did not encounter any difficulties during the experiment	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
No programming experience is required to teach Baxter a new task	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I understood why the generated plan was wrong	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I understood why Baxter failed to complete a task	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Overall, I am satisfied with the ease of completing the tasks in the scenarios	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Overall, I am satisfied with the amount of time it took to complete the tasks	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
If I was a factory worker on an assembly line, it would be easy for me to become skillful at using the system	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

4. What was the most useful part? *

Mark only one oval.

- ☐ Detects types automatically
- ☐ Objects are visualised on the interface
- ☐ Robot learns the action from my demonstration
- ☐ Detects conditions automatically
- ☐ Generate solutions to defined goal automatically
- ☐ Other: _____

5. What part of the programming process did you dislike the most? *

Mark only one oval.

- ☐ Demonstrate an action on Baxter
- ☐ Assign action conditions
- ☐ Create a problem for Baxter to solve
- ☐ Setting a goal
- ☐ Execute a generated plan
- ☐ Other: _____

6. What part of the programming process did you like the best? *

Mark only one oval.

- ☐ Demonstrate an action on Baxter
- ☐ Assign action conditions
- ☐ Create a problem for Baxter to solve
- ☐ Setting a goal
- ☐ Execute a generated plan
- ☐ Other: _____

7. What was the most difficult part of the experiment?

Appendix F

PDDL code used in Chapter 7

F.1 iRoPro planning domain

```
(define (domain iRoPro)
  (:requirements :typing :strips)
  (:types
    element
    position - element
    object - element
    cube - object
    base - object
    roof - object )

  (:predicates
    (clear ?e - element)
    (thin ?e - element)
    (flat ?e - element)
    (on ?obj2 - object ?obj1 - element)
    (stackable ?obj2 - object ?obj1 - element)
  )

  (:action move-vacuum
    :parameters (?obj - object ?fromLoc - element ?toLoc - element)
    :precondition (and (on ?obj ?fromLoc)
      (clear ?toLoc)
      (not(clear ?fromLoc))
      (flat ?obj))
```

```

    (stackable ?obj ?toLoc)
    (clear ?obj) )
:effect (and (on ?obj ?toLoc)
    (not(clear ?toLoc))
    (clear ?fromLoc)
    (not(on ?obj ?fromLoc)) )
)

(:action move-grip
:parameters (?obj - object ?fromLoc - element ?toLoc - element)
:precondition (and (on ?obj ?fromLoc)
    (clear ?toLoc)
    (not(clear ?fromLoc))
    (thin ?obj)
    (stackable ?obj ?toLoc)
    (clear ?obj) )
:effect (and (on ?obj ?toLoc)
    (not(clear ?toLoc))
    (clear ?fromLoc)
    (not(on ?obj ?fromLoc)) )
)

(:action side-pp
:parameters (?obj - object ?fromLoc - element ?toLoc - element)
:precondition (and (on ?obj ?fromLoc)
    (clear ?toLoc)
    (thin ?obj)
    (stackable ?obj ?fromLoc)
    (not(clear ?obj)) )
:effect (and (on ?obj ?toLoc)
    (not(clear ?fromLoc))
    (clear ?fromLoc)
    (not(on ?obj ?fromLoc)) )
)
)

```

F.2 iRoPro planning problems

```
(define (problem study-task3-swap)
(:domain iRoPro)
(:objects
  posA posB posC posM - position
  obj1 obj2 - base)
(:init (clear obj1) (on obj1 posA) (clear obj2) (on obj2 posB)
  (clear posC) (not(clear posB)) (not(clear posA)) (clear posM)
  (stackable obj1 obj2) (stackable obj1 posA) (stackable obj1 posB)
  (stackable obj1 posC) (stackable obj1 posM) (flat obj1)
  (stackable obj2 obj1) (stackable obj2 posA) (stackable obj2 posB)
  (stackable obj2 posC) (stackable obj2 posM) (flat obj2)
  (flat posA) (flat posB) (flat posC) (flat posM))
(:goal (on obj1 posB) (on obj2 posA) )
)

(define (problem blocksworld-task2)
(:domain iRoPro)
(:objects
  posA posB posC posD posE posM - position
  obj1 - base
  obj2 obj3 obj4 - roof)
(:init (clear obj1) (on obj1 posB) (clear obj2) (on obj2 posM)
  (clear obj3) (on obj3 posC) (clear obj4) (on obj4 posE)
  (not(clear posC)) (not(clear posM)) (clear posA) (not(clear posB))
  (clear posD) (not(clear posE)) (stackable obj1 posA)
  (stackable obj1 posB) (stackable obj1 posC) (stackable obj1 posD)
  (stackable obj1 posE) (stackable obj1 posM) (flat obj1)
  (stackable obj2 obj1) (stackable obj2 posA) (stackable obj2 posB)
  (stackable obj2 posC) (stackable obj2 posD) (stackable obj2 posE)
  (stackable obj2 posM) (thin obj2) (stackable obj3 obj1)
  (stackable obj3 posA) (stackable obj3 posB) (stackable obj3 posC)
  (stackable obj3 posD) (stackable obj3 posE) (stackable obj3 posM)
  (thin obj3) (stackable obj4 obj1) (stackable obj4 posA)
  (stackable obj4 posB) (stackable obj4 posC) (stackable obj4 posD)
  (stackable obj4 posE) (stackable obj4 posM) (thin obj4) (flat posA)
  (flat posB) (flat posC) (flat posD) (flat posE) (flat posM))
```

```

(:goal (and (on obj1 posM) (on obj3 obj1)
              (on obj4 obj3) (on obj2 obj4)
)))

(define (problem blocksworld-task34)
  (:domain iRoPro)
  (:objects
    posA posB posC posD posE posM - position
    obj1 obj2 obj3 obj4 - roof)
  (:init (clear obj4) (clear posC) (clear posB) (clear posE)
    (on obj1 obj2) (on obj3 obj1) (on obj4 obj3) (on obj2 posM)
    (stackable obj1 posA) (stackable obj1 posB) (stackable obj1 posC)
    (stackable obj1 posD) (stackable obj1 posE) (stackable obj1 posM)
    (thin obj1) (stackable obj2 posA) (stackable obj2 posB)
    (stackable obj2 posC) (stackable obj2 posD) (stackable obj2 posE)
    (stackable obj2 posM) (thin obj2) (stackable obj3 posA)
    (stackable obj3 posB) (stackable obj3 posC) (stackable obj3 posD)
    (stackable obj3 posE) (stackable obj3 posM) (thin obj3)
    (stackable obj4 posA) (stackable obj4 posB) (stackable obj4 posC)
    (stackable obj4 posD) (stackable obj4 posE) (stackable obj4 posM)
    (thin obj4) (flat posA) (flat posB) (flat posC) (flat posD)
    (flat posE) (flat posM))
  (:goal (and (on obj1 obj2) (on obj3 obj1)
              (on obj4 obj3) (on obj2 posB)
)))

```