From Demonstrations to Skills for High-level Programming of Industrial Robots

Maj Stenmark, Elin A. Topp
Dept of Computer Science, Faculty of Engineering
Lund University, Sweden
e-mail: {maj.stenmark, elin_anna.topp}@cs.lth.se

Abstract

In this paper we describe our approach to robotic skill representation and a prototypical implementation of a programming-by-demonstration approach that allows users to generate skills and robot program primitives for later refinement and re-use. We intend to evaluate the applicability of this approach to high-level programming in a user study, which we also explain.

1 Introduction

In recent years we have had a renaissance in industrial robotics: a new segment of robot models (e.g., UR5, ABB YuMi, Rethink Robotics Baxter and LBR iiwa) intended for safe human-robot collaboration and simpler robot programming have emerged. The application areas for these models are typically assembly of consumer electronics or repackaging of (small) products. The difference between these small robots and previous models is the lead-through programming mode, or kinesthetic teaching, where the user guides the robot arm into the desired positions and thus record and replay a robot program. Dynamic environments, uncertainty and the use of noisy sensor feedback are still challenges for the operator (Stenmark et al. 2016), however, this type of programming tool has enabled industrially oriented research in robot programming by demonstration, PbD, or learning from demonstration, LfD (Billard et al. 2008). Originally, users demonstrate tasks and classic machine learning techniques are used to extract model parameters. Very often the overall qualitative goal is more important than the specific implementation and precision of the execution, see e.g., (Niekum et al. 2015; Ahmadzadeh et al. 2015). For the industrial context, the most obvious drawback with pure data driven approaches is lack of data: it is difficult and time consuming for the user to provide a sufficient number of demonstrations with enough feature variation and precision, that would allow to distinguish between qualitative and quantitative aspects of the task. Hence, the desired approach in an industrial context is a one-shot demonstration, from which enough parameters for an executable representation can be extracted. The representation has to be understandable and adaptable for the user, because the instruction process usually involve multiple iterations in a refinement and testing loop in order to achieve acceptable precision and robustness.

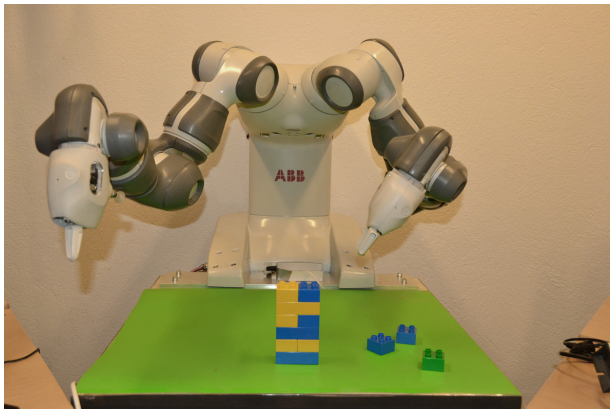

Figure 1: The dual-arm ABB YuMi robot used in the experiments.

Orthogonal to the PbD approach is the use of high-level instructions to simplify robot programming for industrial tasks, e.g., natural language and semantically defined parametrized motion primitives, often called *skills* (Pedersen et al. 2016). Such semantic descriptions are desirable because they enable the automatic generation of, e.g., standard PDDL[1]-descriptions for planning and scheduling of tasks or path planning and generation from virtual models. Of course, extensive modeling requires engineering effort, hence, our uncontroversial hypothesis is that a middle ground should be reached by combining programming by demonstration and parametrized skill representations to simplify robot programming. The novel angle of our research is that the users are to create the parameterized skills from scratch, instead of using a library of expert made skills. The latter approach, while desirable, is currently unattainable since there is no widely adapted architecture in place for experts to create and distribute robot skills. We describe a system that assists the user to combine demonstrations and semantic action information in order to simplify re-use.

We also touch a methodological problem: while there ex-

[1]Planning Domain Definition Language

ist multiple proposed task representation frameworks, evaluations in realistic application settings are rare – but exist, see (Mollard et al. 2015) – and the metrics are chosen by the individual researchers. User studies are often dependent on the laboratory setup and software is seldom shared, not to mention complete systems, hence studies are not easily reproduced by other research groups. We propose thus an approach that uses open-source software components and openly available tools to provide easier robot programming and allows for evaluations in realistic settings.

In this short-paper, we outline an ongoing user study aimed at evaluating the applicability of our previously presented skill representation (Stenmark and Malec 2015) used in a programming by demonstration setup with the ABB YuMi robot, see Fig. 1. First we provide some background about the representation under evaluation, followed by a description of the study setup and evaluation criteria. We discuss the prospective results of the study from a methodological perspective.

2 Skills and high-level programming for industrial robots

A re-usable robot program, in the literature often referred to as a robot skill, has two components. The first is a high-level step-by-step instruction how to achieve a goal. In our work, we represent the robot program using a finite state machine. The state machine can have other skills as nested states and the lowest semantically described step comprises atomic states called *primitives*. The primitives must have a mapping to executable code on the robot system (Stenmark, Malec, and Stolt 2015), which is the second component of the skill[2]. The same execution environment can be used to execute different task state machines but can be very different depending on the application. In our robot system, force-controlled tasks expressed in the iTaSC formalism (De Schutter et al. 2007) use code generated from Simulink models for the force-control (Blomdell et al. 2005), position-based control policies can execute as C-programs that send joint values directly to the robot controller, while other moves execute as native robot code (ABB RAPID).

The primitives have a type hierarchy (Stenmark and Malec 2015). Only the relevant subset of primitives are listed here: the base type is **Action** with subtypes **Motion**, **Gripper Action** and **Locating Action**. Each primitive has a set of semantically annotated parameter types with (system dependent) default values.

Motion has subtypes **Free Motion** and **Contact Motion**. **Free Motion** has subtype **AbsJointMove** (where the target is expressed in joint angles), **Linear Move**, **Circular Move** and **Joint Move** where the target is expressed in Cartesian space relative to a point on an object and the path will be linear, circular or by moving all joints simultaneously. More complex trajectories are expressed as position-based control policies using dynamic movement primitives, DMPs (see Sec. 3). A **Contact Motion** can either be a **Guarded Search** or **Force-controlled Motion**. A **Guarded Search** is a motion towards a point that will continue until contact with an object is reached or, if it fails, a specified target position is reached. A **Force-controlled Motion** has a set of force constraints which have to be fulfilled during the execution, e.g., keeping a constant force in one direction while moving in another. Parameter types to each motion primitive are the target position in a reference coordinate system (when relevant) and error tolerance, velocity and guards.

[2]A forced analogy can be made to Java where a program consists of (byte code generated from) high-level text (re-usable) but the execution is carried out on different virtual machines depending on operating system.

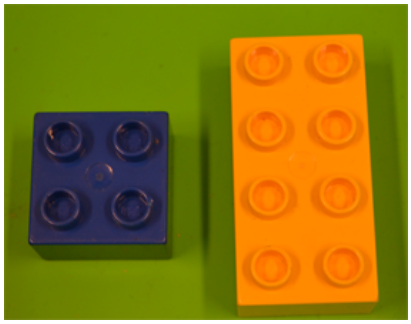

Figure 2: To the left is a 2-by-2 LEGO Duplo piece and to the right a 2-by-4 piece.

Gripper Action has subtype **Open**, **Close**, **Finger Position** with gripping force (and finger positions) as a parameter value. When a suction tool is installed subtypes include **Suction On** and **Suction Off**.

Locating Action has only one subtype and that is **Vision** which locates an object given a calibration and a trained model.

In addition to the primitives, objects in the workspace are specified by the user with types, object coordinate frame and relative points as coordinate frames. The base type for objects involved in an assembly is **Workpiece**. In the study the LEGO Duplo pieces shown in Fig.2 are used, hence, an example of a subtype of **Workpiece** is **Duplo piece** which in turn has subtypes **2-by-2 Duplo** and **2-by-4 Duplo** . The Duplo pieces will have an object coordinate frame in the center of the piece and relative coordinate frames as corners or gripping positions.

A skill can further use control structures such as synchronized motions or synchronized points, loops over objects and conditional branching.

3 Extracting skills from demonstration

Using lead-through (also called kinesthetic teaching), the task can be demonstrated either by recording a sequence of via points and then replaying them in order, or, when necessary, by logging the full trajectory and generalizing it to a position-based control policy. A popular approach for the latter is using dynamic movement primitives, DMPs (Ijspeert et al. 2013), which encode the motion as a nonlinear attractor system towards the goal point with a perturbing force term. The force term is defined as a number of weighted Gaussian basis functions dependent on a phase variable (that scales the motion from 0 to 1 along the trajec-

tory) and the weights are calculated from the demonstrated trajectory. The goal can be changed and the equation will still converge since the forcing function decays with time.

Only one demonstration is needed to create an executable program, but neither approach provides semantic information about the task such as a coordinate system for the motion, the required position accuracy or the high-level goal of the task, e.g., to insert one object on top of another. The parametrization is required for reuse and modification.

In our current implementation, the user will provide (a hypothetical) action type information using speech while moving the robot or by selecting the action type in a touch-based graphical user interface. The system provides initial type suggestions from the log, e.g., if the robot moved, the type hypothesis will be a motion to the new point using the same velocity and position tolerance as the previous move, or a gripper action if the gripper state changed. The user has to actively select the suggestion in order to add it to the sequence, since many motions are unnecessary (e.g., adjustment of the robot elbow to let the user reach around the arm). Some parameter values can be extracted directly using the robot interface, this includes joint angles, the grippers' Cartesian and finger positions, and numerical values for contact forces. It is more difficult to accurately guess the reference coordinate system from a single demonstration, it is more intuitive and thus easier to use speech and a GUI on the other hand. Still the system can suggest points using simple heuristics such as the last used reference coordinate system or, if no such exists, the reference point closest to the tool tip. The log is not discarded so that the user can switch between motion types and parameters without repeating the demonstration. The object positions are taught by pointing using the robot finger tips.

Refinement and debugging are carried out by executing the actions stepwise and adjusting positions and parameter values when needed, or changing, adding or deleting actions.

4 Re-usable skills for high-level programming

When the user has created a task (sequence), where each action has type information and actual parameter values extracted from the demonstration, the task can be saved as a robot skill with type (e.g., **Pick**, **Place** or **Insert**) and a textual description. The task is stored with information about robot system and workspace including object positions and types. When a skill is selected for re-use, the action type information is used to check that the program is valid in the current system, i.e., that the execution environment exists (e.g., there is a module for execution of DMPs), that the parameters are valid (e.g., a target specified using joint angles is only valid for the same robot type) and that object references are selected from the current workspace. The system suggests changes when possible, that is, changing the motion type to a linear move from a DMP and selecting an object with the same super type as reference object. This will give a valid program skeleton that the user can refine using stepwise debugging and updating.

5 Evaluation of re-usable skills in non-expert high-level programming

For the user study, a group of undergrad engineering students in a robotics course will be given the task to program one arm on a dual-arm ABB YuMi to assemble different types of LEGO Duplo bricks. The robot is equipped with simple two-finger grippers, a camera and contact detection, and it provides the previously mentioned lead-through mechanism for task demonstration. The number of participants will be approximately 30 students, expected to have introductory knowledge about robots and programming. They will program the robot using a simplified graphical user interface for high-level sequencing, parametrization and saving of tasks, with limited natural language support for opening and closing the grippers and saving positions.

The programming has two phases: the first phase will be carried out by all participants while the second phase divides the users into three groups. The first group, Group A, will re-use their own skills, Group B will re-use an expert-made skill and Group C is a control group that programs the task without re-use. In phase 1 the users will program the robot from scratch using lead-through and the test application to pick up a 2-by-2 Duplo piece and using insertion to place it on top of a tower of other pieces. In the second phase, a 2-by-4 piece should be picked and placed on the tower in three different positions. Group A will be allowed to parametrize the picking and insertion as skills, Group B will instantiate a similar but expert-made skill and the control group (Group C) has to program the three insertions from scratch as in phase 1. Since the initial program cannot be used directly, the users of Group A and B will have to chose to change either the pick position of the larger piece using lead-through or by inserting parameter values for the coordinates, or by changing the insertion step by adding additional motions/adjusting positions or force. Group B will get a parametrized sequence of primitives that they have to adapt by changing parameter values or adding/deleting actions.

The time to complete the task will be limited and the quality of the resulting program will be evaluated using the success rate of 20 executions. Additional metrics include the number of test cycles the user has to execute (number of parameters changed) or actions added and how many times the users request help. In the second phase we aim to understand whether the users can finish before the maximum allowed time runs out. The first phase is not intended to take more than 10 minutes, while the second will be limited to 30 minutes. The result is intended to give an indication whether or not parametrization and re-use reduce (and in that case, how much) the programming effort in similar tasks of moderate complexity.

The explorative aspects of the study will show whether or not the users preferred to change parameters and/or refine the action sequence, whether or not they preferred language commands or graphical interaction only as well as reveal any common problems non-expert users have with robot programming that must be addressed during training.

However, general conclusions will be difficult to draw.

The result of the study is obviously dependent on the usability and quality of the test application, the responsiveness of the language detection and other quality aspects of the tools used, however, the application will be open source and stand alone to simplify comparisons. Secondly, the task of assembling LEGO bricks is a rather specific task, and it might not be entirely representative for other assembly tasks, although we assume the necessary components for inserting one brick into another to be quite similar to, e.g., folding insertion operations in mobile phone assembly.

6 Discussion

This paper has provided an overview of an ongoing study to evaluate the usability of a robot programming approach where the operators instruct the robot using lead-through and parametrized primitives to create their own re-usable robot skills from scratch. The study aims at evaluating if such parametrization will lower the programming effort when adapting the robot program to a similar task.

The result of the study is essential when deciding the direction of future research efforts. Our initial hypothesis is that users programming robots for industrial assembly task can benefit significantly from a combination of task demonstration and skill parametrization based high-level approaches. However, it might also be the case that it is easier or more efficient to rely fully on direct task demonstration also for repeated programming of similar tasks, which can be due to different factors. Comparisons between the groups can be made, i.e., is it beneficial for the user to know about the parametrization and execution sequence when adapting the skill? Is it easier for the users to program everything from scratch than try to decipher someone else's program?

Hence, *if* the results of our study show that the skill parametrization does *not* simplify the programming of similar task by reducing programming time or success rate of the application, this can be due to the taught skills being too simple to begin with and that the overhead of creating a reusable skill is too high compared to the skill complexity and even that the relationship between skill complexity and best approach for programming is non-linear. This can in turn depend on the Duplo building task or the quality and usability of the study tool application. Further evaluations and investigations of the reasons for such an outcome would have to be considered. *If*, on the other hand, the programming effort is reduced, further investigations in similar setups would be needed to ensure that our approach and representations are not simply overfit to Duplo building while ill suited as general benchmarking.

This is artifact research and while action models and frameworks are well motivated from a qualitative perspective, it is very challenging to create experiments from which valuable results for a transfer into real-world applications can be drawn. There is a lack of benchmarking and methodological tools for system evaluation which holds back progress in the field of AI-HRI because the ideas cannot propagate from academia to industry without measurable value. We believe, that our study is one contribution to a methodological discussion that is necessary to establish this type of tools.

7 Acknowledgments

The research leading to these results has received funding from the European Community's Framework Programme Horizon 2020 under grant agreement No 644938 SARAFun.

References

Ahmadzadeh, S.; Paikan, A.; Mastrogiovanni, F.; Natale, L.; Kormushev, P.; and Caldwell, D. 2015. Learning symbolic representations of actions from human demonstrations. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. 3801–3808.

Billard, A.; Calinon, S.; Dillmann, R.; and Schaal, S. 2008. Robot programming by demonstration. In *Springer handbook of robotics*. Springer. 1371–1394.

Blomdell, A.; Bolmsjö, G.; Brogårdh, T.; Cederberg, P.; Isaksson, M.; Johansson, R.; Haage, M.; Nilsson, K.; Olsson, M.; Olsson, T.; et al. 2005. Extending an industrial robot controller-implementation and applications of a fast open sensor interface. *IEEE Robotics & Automation Magazine* 12(3):85–94.

De Schutter, J.; De Laet, T.; Rutgeerts, J.; Decré, W.; Smits, R.; Aertbeliën, E.; Claes, K.; and Bruyninckx, H. 2007. Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty. *The International Journal of Robotics Research* 26(5):433–455.

Ijspeert, A. J.; Nakanishi, J.; Hoffmann, H.; Pastor, P.; and Schaal, S. 2013. Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Comput.* 25(2):328–373.

Mollard, Y.; Munzer, T.; Baisero, A.; Toussaint, M.; and Lopes, M. 2015. Robot programming from demonstration, feedback and transfer. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, 1825–1831.

Niekum, S.; Osentoski, S.; Konidaris, G.; Chitta, S.; Marthi, B.; and Barto, A. G. 2015. Learning grounded finite-state representations from unstructured demonstrations. *The International Journal of Robotics Research* 34(2):131–157.

Pedersen, M. R.; Nalpantidis, L.; Andersen, R. S.; Schou, C.; Bgh, S.; Krger, V.; and Madsen, O. 2016. Robot skills for manufacturing: From concept to industrial deployment. *Robotics and Computer-Integrated Manufacturing* 37:282 – 291.

Stenmark, M., and Malec, J. 2015. Knowledge-Based Instruction of Manipulation Tasks for Industrial Robotics. *Robotics and Computer Integrated Manufacturing* 33:56–67.

Stenmark, M.; Stolt, A.; Topp, E. A.; Haage, M.; Robertsson, A.; Nilsson, K.; and Johansson, R. 2016. The GiftWrapper: Programming a Dual-Arm Robot With Lead-through. In *ICRA Workshop on Human-Robot Interfaces for Enhanced Physical Interactions*.

Stenmark, M.; Malec, J.; and Stolt, A. 2015. From high-level task descriptions to executable robot code. In *Intelligent Systems' 2014*. Springer. 189–202.