In: Proceedings of the 24th International Symposium on Industrial Robots (ISIR '94)

Robot Programming by Demonstration (RPD) - Using Machine Learning and User Interaction Methods for the Development of Easy and Comfortable Robot Programming Systems

S. MÜNCH, J. KREUZIGER, M. KAISER, R. DILLMANN

University of Karlsruhe
Institute for Real-Time Computer Systems and Robotics
Kaiserstr. 12
D-76128 Karlsruhe
GERMANY

Abstract - *Robot Programming by Demonstration is an intuitive method to program a robot. The programmer shows how a particular task is performed, using an interface device that allows the measurement and recording of the human's motions and other parameters that are relevant to perform the demonstrated task. This paper presents an analysis of the learning and interaction requirements that are characteristic for an RPD system. Based on these requirements, a new system architecture is proposed that supports all phases of the interactive programming process. For an example task, experimental results are given.*

Keywords: Programming by Demonstration, Man-Machine Interface, Machine Learning

1. INTRODUCTION

Due to the costs involved in the development and maintenance of robot programs, automatic programming techniques and Programming by (Human) Demonstration (PbD) are currently attracting a lot of interest (e. g. [8], [15], [18]).

Moreover, learning capabilities are becoming continuously attractive for robotic researchers, and Machine Learning techniques are applied on several levels of robot planning and control.

Integrated architectures that contain both the learning and the performance component are a topic of ongoing research (e.g. [10], [14]). It is quite obvious that both developments can be seen as closely interrelated, since learning capabilities are required to increase the robot's ability to interact with a human operator as well as to perform operations autonomously. On the other hand, the use of interaction and the (implicit) knowledge of the operator can help guiding the learning processes in order to achieve satisfactory results.

This paper describes the basic requirements concerning the application of Machine Learning and the role of user interaction in an RPD system in the light of the task an RPD system has to solve.

The next section gives a brief overview on RPD, whereas section 3 deals with the role of Machine Learning and user interaction in an RPD system. Based on the requirements analysis, an architecture for an RPD system is proposed in section 4. Section 5 describes experiments with a prototypical implementation, showing the validity of the proposed concepts.

2. ROBOT PROGRAMMING BY DEMONSTRATION

The idea of Programming by Demonstration (PbD) is to specify task solutions or to describe knowledge in general not by explicitly programming each detail, but by demonstrating several examples of a task or problem solution. This leads to a much easier and more natural way of adapting an automatic system of any kind to the user's needs. In fact, the paradigm of PbD has already been investigated for user-interfaces [7], graphical editors [17], and office automation systems [2]. In each case the system acts as an observer of a human performing tasks and tries to learn from these observations and/or dialogs with the user. Thus, such a user-adaptive system can learn typical action sequences from the user and try to apply them autonomously if it encounters a similar situation the next time.

Applied to robotics, PbD can be a considerable improvement for making robot programming feasible even for beginners and easy and comfortable for experts. Similar to traditional robot programming, several different methods can be used for RPD:

1. The user demonstrates a solution by commanding the robot (by 6D-bowl, master-slave approach, teleoperation, or by moving the real robot at hand);
2. The user demonstrates a solution by performing the task manually - without a robot [9, 21].

In both cases virtual reality methods might also be used.

Though the second method seems to be the most natural way of programming, it has a couple of drawbacks. Especially the transformation of human motions to adequate robot motions is difficult or even impossible, due to kinematic constraints of the robot and the enormous manipulation skills of a human. In contrast, the first method enables the system to exactly record the demonstrated solution, which makes a transformation into 'robot capabilities' superfluous and is necessary for skill learning.

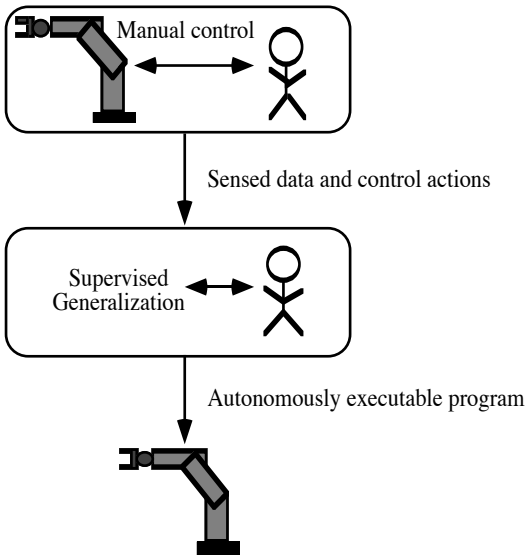

Fig. 1: The user's role in RPD

In general, RPD releaves the user, acting as a programmer or teacher, from time-consuming nit-picking step-by-step programming. It pushes the task of robot programming to a higher conceptual level. The interaction between the user and the system, however, is a very important issue. As long as the preprocessing, generalization, and spezialization techniques cannot be applied fully autonomously, the system must rely on the support and the 'good-will' of the user.

3. THE ROLE OF ML AND USER INTERACTION IN AN RPD SYSTEM

3.1 Machine Learning

The degree of learning as it is necessary to fulfill the previously described tasks can reach from simple rote learning (i. e. just remembering exactly what the user did) to complex learning methods using inductive generalization and specialization or deductive approaches. Whereas the possibilities and limitations of simply recording a solution and playing it back the same way have been investigated for years in standard teach-in methods for robots, only few efforts have been made to apply real learning techniques [12, 13].

As the approach in this paper has been mainly developed with industrial users in mind, one strong demand was the readability and comprehensibility of the resulting operational knowledge. Therefore, the resulting knowledge is represented as a generalized explicit program. In these programs elementary skills are considered as black boxes which are referenced inside the program steps.

For the task of learning from demonstrations (i. e. examples), a lot of work is known in the ML literature. Nevertheless, most of it cannot be simply applied to the task at hand. The specific learning tasks can be described as follows:

1. Learning of structural knowledge (program structure, macro operators / subprograms);
2. Learning of parameter knowledge (termination criteria, condition parameters, action parameters);
3. Learning of skill knowledge.

Given a set of traces of user demonstrations which have been segmented into lists of elementary operations (EOs), one major task is to determine a general program structure. For that purpose one single sequence of EOs is analyzed whether certain subsequences appear repeatedly. These are candidates for generating a new *macro operator*. Additionally, loops can be searched for [8], so that a repeating sequence of EOs and/or macro operators can be replaced by an explicit loop construct.

The task of *structural induction* is the 'integration' of several example structures as described above into one general structure using conditional branches [5]. On this level, the task is mainly a matching problem, i. e. to find the correspondence between EOs and/or macro operators of one sequence with actions in the other sequences.

The main problem, however, is that of *learning parameter descriptions*. For each branch and loop condition and for each EO, the dependency of the condition resp. action parameters on the task or on example specific features must be determined. This problem is known in ML as functional induction, but so far no really satisfying solutions have been found. Most often a generate-and-test approach is used, such as generating all possible combinations in the function space and checking their applicability [8].

The functional induction task in RPD is more difficult than those investigated in most ML approaches [6, 16], because the set of relevant variables in the world model or the task description, that should be part of the function description, is not explicitly given. One possibility to simplify that problem would be a user interaction as described in the next section.

Further application of the PbD paradigm in the area of robotics concerns the acquisition of elementary skills. These elementary skills must be provided by the robot for the execution of elementary operations. They represent the interface be-

tween the planning and the control level in the robot's architecture. Similarly, Machine Learning techniques supporting their acquisition and representation are located at the interface between subsymbolic and symbolic learning.

While neural networks and fuzzy controllers have already been established as feasible alternatives to classical control paradigms throughout a number of applications, the generation of these controllers is an unsolved problem. Therefore, the idea of applying a generalization technique on a set of examples in order to arrive at an operational controller for the demonstrated elementary operation is very appealing.

However, even if the task seems conceptually similar to the generation of robot programs from examples on a higher level, there exist a lot of practical differences. The first and probably greatest arises from the nature of the elementary operations. Instead of a sequence of discrete actions, the control involved in the supervision of an elementary operation (e. g., a force-controlled movement, contour-tracking, etc.) requires the learning of a continuous numerical function [1, 3, 11].

Therefore, completeness or sufficiency of the examples is much more important, but much more difficult to achieve than on the task level.

Even bigger problems arise from the nonoptimal input provided by the user. While this can relatively easily be detected by the user himself on the task level, it is almost impossible to classify single control inputs as good or bad. Usually, only the complete execution can be regarded as either good or bad. To assign this delayed evaluation to specific parts of the control action is the classical problem of reinforcement-learning resp. learning with delayed rewards [20, 22].

Because of the problems described above, until now only preliminary results in this area exist. They range from the initial generation of a neural network or a fuzzy controller on the base of some existing controller and later on-line tuning on the field over the interpretation of the whole control task as a markovian model to the direct playback of a compliance trajectory [3, 11, 22, 23]. In each of these approaches a lot of simplifying assumptions that do not hold in reality are made. A possible way to overcome these problems might be a better integration of the user into the acquistion system, not only by allowing him to supervise the whole process, but especially by providing him with new means to analyze and evaluate the examples given and the results produced.

3.2 User Interaction

An important aspect concerning the basic idea, but also the practical realization and the usability of an RPD system, is the interaction of the user, acting as a teacher, trainer, or programmer, with the system. The complete programming process, from the generation and supplement of examples to the evaluation of the resulting robot program, can be subject of user control.

Following the structure of this programming process, the first task assigned to the user is to provide the system with examples (*example generation*) of possible solutions for a given problem. An important constraint is the necessity to give examples in a form that can be understood by the RPD system *and* executed by the robot. Therefore, the user must know how the robot can solve the given task and must be able to demonstrate the solution without implicitly using any resources the robot does not have. If, for instance, the user detects a contact situation visually, and the robot is only equipped with a force/torque sensor, the user must take the robot's perception into account. Similarly, the user is responsible for providing a sufficient set of examples to achieve a desired generality.

Example interpretation must also deal with another aspect: The system itself does not know anything about the quality of the given examples. The control inputs and the sensory feedbacks it recorded might be optimal for a given configuration, they can represent a possible but inefficient solution, or they even result from an incorrect demonstration. The user must therefore classify examples according to their quality.

The *preprocessing of examples* in order to generate a normalized representation in terms of the existing elementary operations requires user support, too. The RPD system has no evidence concerning the logical connection of several data it recorded during a demonstration. Only the user can partition the data trace, yielding segments that can be identified with single elementary operations.

During the generalization phase, the user must *guide the learning process*. The RPD system itself cannot decide whether a recorded action represents an important part of the demonstration, a correction of a previously encountered error, or just an inefficiency introduced by the user. It is also not able to distinguish between task-specific parameterization and example-specific parameterization. Both constraints require the user to instruct the system where to generalize and where to keep the existing instantiation also for the general solution that is to be stored.

At the end of the learning process, the RPD system has produced a description of a general solution for the given task. An *evaluation of this learning result* has to be done by the user. Possible deficiencies in the solution should be detected and improved by repeating parts of the processing chain or, if necessary, providing additional examples for previously uncovered regions of the input domain.

The final output of the RPD system must be directly executable by the robot, such as, for instance, a robot program, or a sequence of parameterized elementary operations that remain under the control of the system. Provided that a good set of examples was available and the learning system could produce an optimal general solution, the instantiation of this general solution should yield a near optimal executable program. In practice, however, *tuning of a generated program* by the user could require little effort compared to providing a complete and optimal set of examples.

Summarizing these observations, it is obvious that the interaction of user and system plays a key role in the concept of RPD. During all phases of the programming process, the user remains the highest authority. The ultimate target is of course an autonomous RPD system, taking examples as input and producing program templates and executable code as output. However, as long as limited sensorial and reasoning capabilities exist, the supervision and guidance of all phases of the programming process by the user will remain necessary.

4. RPD SYSTEM ARCHITECTURE

In order to realize an RPD system, the different requirements arising from the individual phases of the programming process must be taken into account. They specify the functionality of and the interfaces between the components of an RPD system and are therefore the main criteria for the design of the system's architecture.

Figure 2 shows our proposed solution for an RPD system's architecture, meeting the requirements described below. A prototypical implementation, based upon this architecture, has been realized and used for the experiments described in the next section.

The programming process starts with the demonstration. The user can supply examples for a specific task in several ways, for instance by directly moving the robot along a desired path, or by means of an *input device* such as a 6D-joystick. During each of the demonstrations, the values of the significant parameters (such as position, velocity, tool status, etc.) are recorded, such that an example becomes a set of parameter traces over time. These parameter traces can also be generated artificially, if a sufficiently accurate simulation exists. However, the simulation usually hides important aspects of the real environment.

Further processing of the given examples on a cognitive level requires a normalized representation that does not exhibit unnecessary details of the individual demonstration. This representation is based on the set of elementary operations that is provided by the robot. Aided by the user, an *analyzer* scans each of the demonstrations and generates the corresponding sequences of elementary operations as well as the necessary control structures. In order to perform this operation as efficient and accurate as possible, the analyzer and also the induction unit must have access to a *world model*.

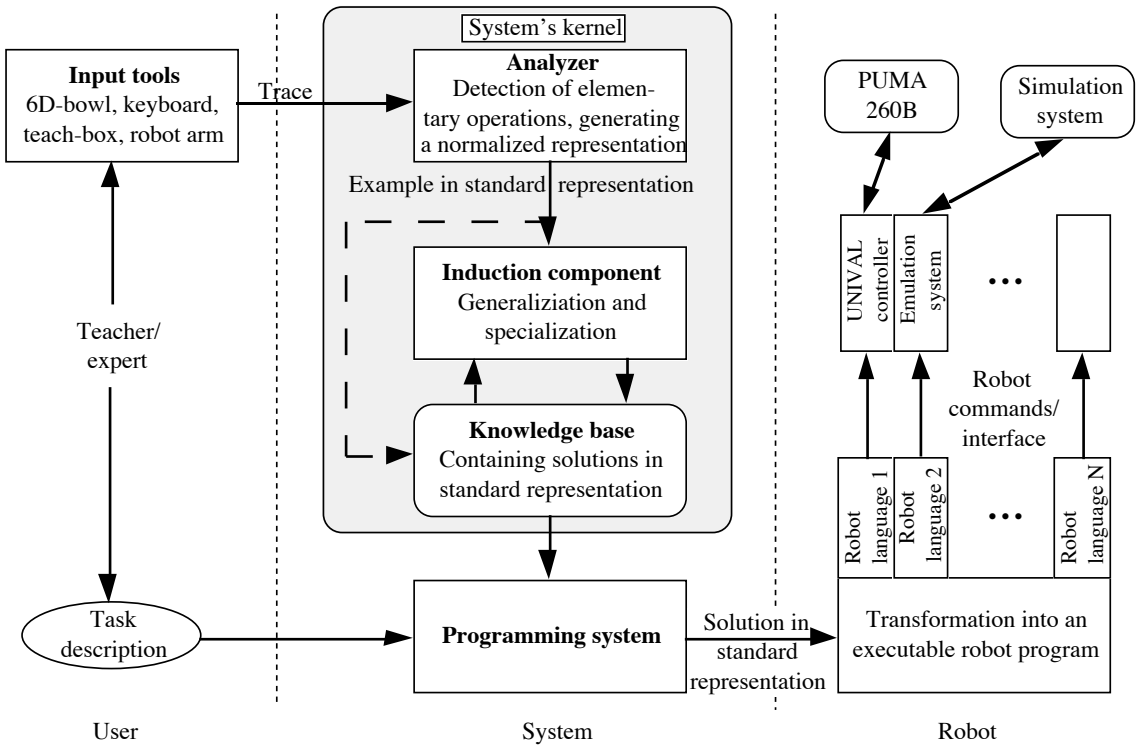

Fig. 2: Functional architecture of the RPD system

An *induction unit* operates on this normalized representation of the examples and produces a generalized description covering all of them. This general solution for the demonstrated kind of task is stored in the RPD system's *knowledge base* and can be used later, both for the generation of executable programs and for the processing of new examples. User support is also necessary during this phase, e. g., for classifying individual actions according to their importance and for guiding the generalization process.

The generation of an executable program requires the instantiation of the general solution according to the needs of the real task. For instance, variable locations must be replaced by exact locations, and for sensor-guided actions appropriate limits and error detection measures must be found. A possible way to achieve this task is to use a simulation of both the world, the robot, and the necessary sensors. However, the accuracy of the simulation is equally critical as in the demonstration phase. Therefore, user supervision is also required here.

4.1 Learning Phase

The first thing to do when working with an RPD system is the *system's initialization* with a world model, without which no task processing or learning can be done. Because no standard solution for the desired task is present in the knowledge base in this phase, the learning process starts with the *demonstration* of one example; more examples may follow later.

During the *analysis*, which follows the demonstration, the points relevant for the robot's movement will be extracted, whereas redundant points will be eliminated to reduce the amount of data. The trace constructed that way has to be divided into separate sections. With this partitioning process, the preprocessing of the data is finished. Now each partition of the trace, which is already in a normalized format, can be seen as an elementary operation. The *matching* between the trace data and the EO (that is known to the system) is supported by the user, but can in principle also be done by the system itself autonomously.

The partitioned and pre-interpreted example has to be transformed into a *general format* which allows the processing not only of this single example but of a whole class of similar tasks. The number of examples needed for such a generalization depends strongly on the complexity of the task. The process of generalization and specialization can itself be divided into two different, but overlapping phases: The *structuring* of the solution and the processing of the single's steps parameters.

At first, the system tries to identify sequences of EOs and creates *macro operators* out of it. If a sequence is found which occurs more often than a preset threshold, the user will be asked whether this sequence should become a new macro or not. Only if he agrees, the system will replace all occurences of the sequence by the macro. This process alters the structure of the solution, but keeps the parameters of the commands unchanged. The next step to be processed is the *detection of branches and loops*. In this phase of the generalization process, the parameters have to be taken into account, e. g. to determine the condition for the end of a loop or for selecting one of two distinct branches.

After this phase, the former linear data sequence has been transformed into a *structured tree*. If the solution for a single demonstration has been structured completely, the solutions of two or more examples can be matched in order to find a *generalized solution* which is capable of solving all the tasks covered by the given examples.

The *generalization and specialization* during the matching is based upon common features of or distinctions between different solutions for the same task. The main method used in generalization and specialization of condition and action parameters is *functional induction*. At the moment all object features are used as input variables, and a functional description for the condition resp. action parameter is searched for. So far a complete construction of the possible function space is performed and the best approximating function is used. In a future system that deals with more complex tasks, user interaction is necessary in order to reduce the set of input variables and the dimensionality of the function space.

The whole process of demonstration, analysis, and induction is shown in figure 3. The result of the generalization and specialization is a kind of program called *template,* which is a representative for a whole class of tasks. It has to be integrated in the knowledge base to be used later for similar tasks.

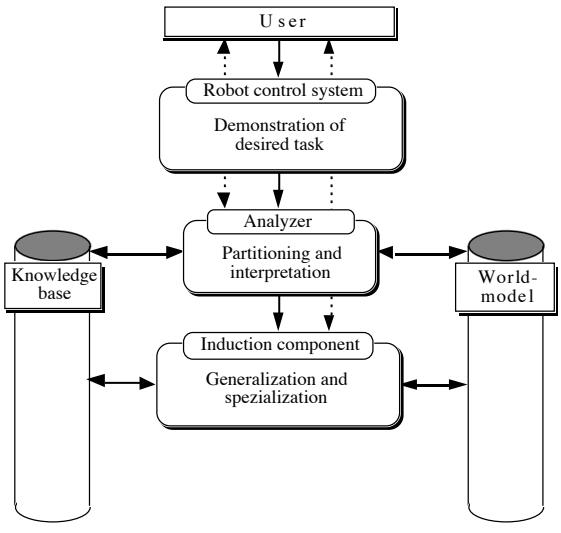

Fig. 3: Schematic description of a demonstration

4.2 Execution Phase

In the second phase, the user defines the desired task, initializes the task specific parameters, and gets a program template. This template, generated during the first phase, is not an executable program but lacks some data which have to be taken from the world model or which have to be provided by the user. After these gaps have been filled, the program is complete. Before it can be executed, it has to be transformed from the RPD system's internal representation into executable code. This transformation is done by an interpreter which is not part of the kernel system.

Finally, the RPD system has produced an executable robot program in a programming language supported by the interpreter. This explicit program can be checked, verified, and, if necessary, changed, extended, or optimized afterwards. The execution phase is summarized in figure 4.

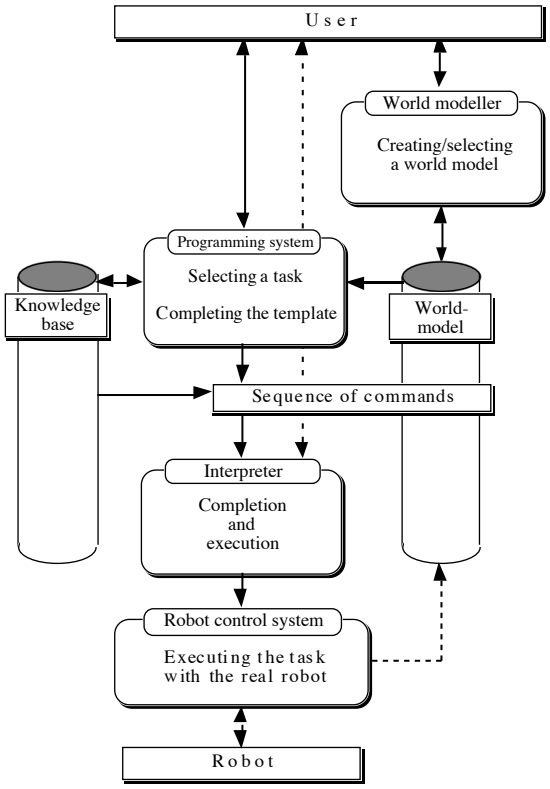

Fig. 4: Schematic description of an execution

5. EXPERIMENTAL RESULTS

A first prototypical RPD system has already been implemented [4]. This section summarizes an example session with the following task: Twelve objects of four different types - small and large cubic-boxes and small and large cylinders - are placed randomly in the shelves of a case. One possible start situation is shown in figure 5.

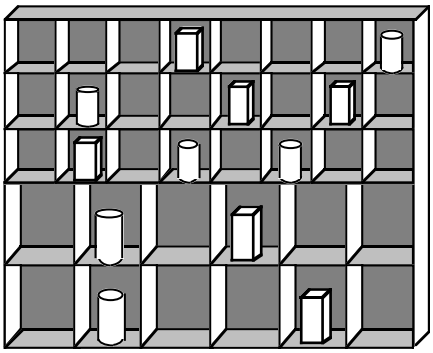

Fig. 5: Start situation of the example

The task is to sort the objects under the constraint that they are placed from left to right, from top to bottom, small objects in small shelves and large objects in large shelves, and that cubic-boxes are placed before cylinders. The goal situation for this task is shown in figure 6.

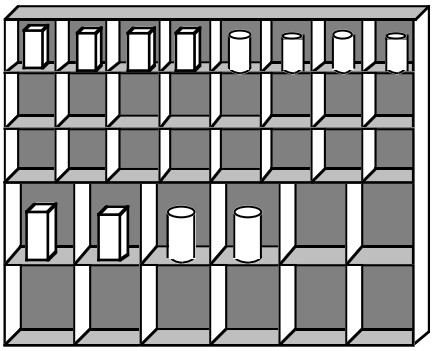

Fig. 6: Goal situation of the example

This task has been chosen because of it's complexity which allows to prove the principle correctness of our approach and to test and improve the system's features and capabilities. It can be extended in several ways when the system's capabilities will allow to handle more difficult tasks, e. g., new objects can be introduced, different ways of sorting following several criteria can be learned, etc.

5.1. Modeling and Demonstration of the Task

After the task has been selected, the world model has to be generated, i. e. the objects have to be modeled and placed into the shelves of the case. This model can be saved and restored to be used for more than one demonstration, of course. When the model has been created, the user must demonstrate a solution for the specified task. In order to write a trace of all actions which he performs with the robot, i. e. moving the manipulator or opening and closing the gripper, he first has to start a timer, thus initializing the recording of the selected data. Depending on the task to perform and the user's experience, the sample rate of the timer can be set to a desired value. Usually, one sample per second yields good results.

As mentioned above, there are several ways to demonstrate a task to a robot. For this example, a 6D-bowl has been chosen, because it allows precise and easy control of the robot's arm. A small part of a trace recorded during the experiment described above is shown below. It includes the values of the 6D-bowl and the world frame of the robot's TCP, but other values (e. g. the joint positions, forces and torques) can be stored as well.

```
((0 0 0 0 0 0 1)(232.850000 475.590000
127.940000 269.989996 1.630000 180.260285))
((0 0 0 0 0 0 0)(232.870000 475.590000
127.970000 270.001010 1.630000 180.250000))
((0 0 0 0 0 0 0)(232.830000 475.590000
127.970000 270.001010 1.620000 180.250000))
((0 0 0 0 0 0 0)(232.900000 437.760000
127.990000 270.003001 1.630000 180.229915))
((0 0 0 0 0 0 0)(232.810000 437.870000
128.000000 270.000000 1.610000 180.240000))
((-127 0 0 0 0 0 0)(200.500000 438.040000
127.740000 269.959989 1.370000 180.130957))
```

5.2. Analyzing the Trace

In the next step, the trace is filtered, transformed, and loaded into the RPD system. Several thresholds can be set to influence this process, e. g. the filtering of 'zero-movements' depends on the sample rate of the tracer. During the transformation process, the data from the trace are compared to the positions of the objects as described in the world model, especially if key events like 'open-gripper' or 'close-gripper' occur. The parameters of the commands building the solution are altered according to the position of objects that can be obtained from the world model. The result will be a list of elementary operations defined in the system's standard description language which can be used for further processing. A small section of this list (without parameters) is shown below:

```
MOVET-REL
SET-GRIPPER
MOVES-REL
MOVET-REL
MOVES-ABS
MOVES-ABS
MOVET-REL
MOVES-REL
SET-GRIPPER
MOVET-REL
```

The structuring of this list is currently done by establishing macro operators instead of EOs. This process has already been described in section 4, but one aspect should be mentioned with regards to the list of EOs and the following figure which represents a structured tree of the same commands: The creation of all macros has been *proposed* by the system but *had to be accepted* by the user, i. e. the creation of macros is not deterministically but can be guided by each user according to his specific needs.

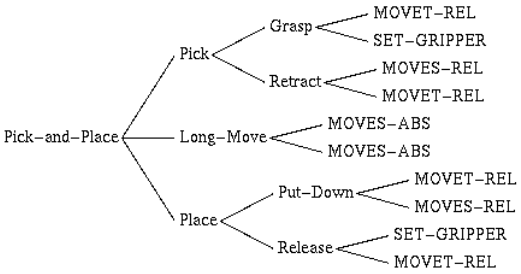

Fig. 7: Solution after generation of macros

5.3. Executing the Solution

The solution tree can now be shown on the screen, saved into a file, or executed on the robot. For the execution, three modes are available which allow testing, real execution, or both. The parameters used in the executable commands are partly taken from the trace and partly from the world model, depending on the respective commands. An output produced in test-mode, representing the above sequence with its parameters, is shown below:

```
(PEO-movet-rel 0 0 85 0 0 0)
(PEO-set-gripper 3800 20)
(PEO-moves-rel 0 0 -10 0 0 0)
(PEO-movet-rel 0 0 -61 0 0 0)
(PEO-movew-abs -67 415 97 180 0 90)
(PEO-movew-abs -35 415 97 180 0 90)
(PEO-movew-abs 127 415 97 180 0 90)
(PEO-movew-abs 227 434 106 180 0 90)
(PEO-movet-rel 0 0 61 0 0 0)
(PEO-moves-rel 0 0 9 0 0 0)
(PEO-set-gripper 4300 -20)
(PEO-movet-rel 0 0 -38 0 0 0)
```

6. CONCLUSION AND FUTURE WORK

The application of Programming by Demonstration (PbD) in Robotics, resulting in Robot Programming by Demonstration (RPD), can be expected to lower the costs of robot programming significantly whilst increasing the flexibility and portability of robot programs. The integration of machine learning techniques and the integration of the user by interaction push the actual programming on a higher conceptual level. However, as long as the steps necessary to transform a user-given example into a good template suitable for further processing and execution cannot be performed fully autonomously, the user will remain the central part of the concept.

The work to be done will therefore concentrate on the automatization of the preprocessing and the generalization steps. New methodologies to support the automatic analysis of given examples, and a suitable feedback between the performance component (the robot and its elementary skills) and the programming system will be investigated.

ACKNOWLEDGEMENTS

This work has been supported by the Siemens AG, München, Germany. We would like to thank Dr. G. Lawitzky and Dr. S. Bocionek for their kind cooperation. The prototype system has been developed at the University of Karlsruhe, Institute for Real-Time Computer Systems and Robotics.

REFERENCES

[1] C. Archibald, E. Petriu. Computational Paradigm for Creating and Executing Sensor-based Robot Skills. In: *Proc. of the 24th Int. Symp. on Industrial Robots [ISIR '93].* Tokyo, 1993.

[2] S. Bocionek, T. M. Mitchell. Office Automation Systems that are 'Programmed' by their Users. In: *Proc. of the 23rd Annual Conf. of the German Association of Computer Science*. Dresden, Germany, 1993.

[3] M. Botta, A. Giordana. SMART+: A multi-strategy learning tool. In: *Proc. of the Int. Joint Conference on Artificial Intelligence (IJCAI '33]*, Chambery, France, 1993

[4] R. Dillmann, S. Münch, J. Kreuziger. *Programming by Demonstration in Robotics: A Study*. Universität Karlsruhe, Institut für Prozeßrechentechnik und Robotik. Karlsruhe, 1993. (In German)

[5] B. Dufay, J.-C. Latombe. An approach to automatic robot programming based on inductive learning. In: M. Brady, R. Paul, editors, *Proc. Robotics Research: The First International Symposium*, S. 97-115. MIT Press, 1984.

[6] B.C. Falkenhainer, R.S. Michalski. Integrating Quantitative and Qualitative Discovery in the ABACUS System. In: Y. Kodratoff et al. (eds.) *Machine Learning - An Artificial Intelligence Approach*, Vol. 3, Morgan Kaufmann Publishers, 1990.

[7] H.-W. Hein, G. M. Kellermann, C. G. Thomas. X-AiD: A Shell for Building Highly Interactive And Adaptive User Interfaces. In: *Proc. of the Int. Joint Conference on Artificial Intelligence (IJCAI '87)*. Mailand, 1987.

[8] R. Heise. *Demonstration instead of programming: Focussing attention in robot task acquisition*. Research Report Nr. 89/360/22, Department of Computer Science. University of Calgary, 1989.

[9] K. Ikeuchi, T. Suehiro. Towards an Assembly Plan from Observation. Part1. In: *Proc. of the 1992 IEEE Int. Conf. on Robotics and Automation*. Nice, 1992.

[10] M. Kaiser, F. Wallner. Using Machine Learning for Enhancing Mobile Robot's Skills. In: *Proceedings of the IRTICS '93*, Madrid, Spain.

[11] M. Kaiser. Using Time-Delay Neural Networks for Robot Control. Submitted to: *Symposium on Robot Control 1994, (SYROCO '94),* Capri, Italy.

[12] J. Kreuziger. Application of Machine Learning to Robotics - An Analysis. In: *Proc. of the 2nd Int. Conf. on Automation, Robotics and Computer Vision*. Singapore, P. RO-15.4.1-RO-15.4.5, 1992.

[13] J. Kreuziger und S. Cord. *Applications of Symbolic Learning Methods in Robotics*. Bericht der Fakultät für Informatik 23/92. Universität Karlsruhe, 1992. (In German)

[14] J. Kreuziger und M. Hauser. A new system architecture for applying symbolic learning techniques to robot manipulation. In: *Proc. of the IEEE/RSJ Conference on Intelligent Robots and Systems*, Yokohama, 1993.

[15] Y. Kuniyoshi, M. Inaba, H. Inoue. Seeing, Understanding and Doing Human Task. In: *Proc. of the 1992 IEEE Int. Conf. on Robotics and Automation*. Nice, 1992.

[16] P. Langley et al. The Search for Regularity: Four Aspects of Scientific Discovery. In: R.S. Michalski et al. (eds.). *Machine Learning - An AI Approach,* Vol. 2, Morgan Kaufmann Publishers, 1986.

[17] M. Sassin, S. Bocionek. Programming by Demonstration: A Basis for Auto-Customizable Workstation Software. In: *Proc. of the Workshop Intelligent Workstations for Professionals*. H. Schwärtzel (ed.). München, Springer, 1992.

[18] G.M. Scott, J.W. Shavlik, W.H. Ray. Refining PID controller using neural networks. In: *Advances in Neural Information Processing Systems 4*, Denver, 1991

[19] A. M. Segre. *Machine Learning of Robot Assembly Plans*. Kluwer Academic Publishers, 1988.

[20] R.S. Sutton. Learning to predict by methods of temporal difference. In: *Machine Learning*, Vol. 3, pp. 9-44, 1988

[21] A. Ude, R. Dillmann. Trajectory Reconstruction from Stereo Image Sequences for Teaching Robot Paths. In: *Proc. of the 24th Int. Symp. on Industrial Robots [ISIR '93]*, pp. 407 - 414, Tokyo, 1993.

[22] C.J.C.H. Watkins. *Learning with delayed rewards*..PhD Thesis, University of Cambridge, 1989

[23] J. Yang, Y. Xu, C.S. Chen. Hidden Markov Model approach to skill learning and its application to telerobotics. In: *Proc. of the 1993 IEEE Int. Conf. on Robotics and Automation*, Atlanta, Georgia, 1993