Robot Programming by Demonstration (RPD): Supporting the Induction by Human Interaction

H. FRIEDRICH, S. MÜNCH AND R. DILLMANN {friedric|smuench|dillmann}@ira.uka.de
*University of Karlsruhe, Institute for Real-Time Computer Systems and Robotics, Kaiserstr. 12,
D-76128 Karlsruhe, Germany*

S. BOCIONEK AND M. SASSIN {bocionek|sassin}@zfe.siemens.de
Siemens AG, Corporate R&D, ZFE ST SN 6, Otto-Hahn-Ring 6, D-81739 Munich, Germany

Editors: Judy A. Franklin, Tom M. Mitchell, and Sebastian Thrun

Abstract. Programming by Demonstration (PbD) is a programming method that allows software developers to add new functionalities to a system by simply *showing* them in the form of few examples. In the robotics domain it has the potential to reduce the amount of time required for programming and also to make programming more “natural”. Just imagine the task of assembling a torch by a manipulator. Wouldn’t it be nice to just assemble the torch with one’s own hands, watched by video and laser cameras and maybe wearing data gloves, i. e. sensors that provide the data to automatically generate the necessary robot program for the assembly task? And wouldn’t it be even nicer to demonstrate the task with few different torches, but achieving an assembly function for all possible variants of them?

In order to realize such a PbD environment, at least two major problems have to be solved. First, the sensor data have to be transformed to high-level situation-action descriptors, a task that is not yet solved in general. Second, if a generalization is required, induction algorithms must be applied to the recorded and transformed traces, aiming to find the most general user-intended function from only few examples. In this article we will concentrate only on the second problem. The described experimental environment consists of an industrial robot (PUMA 260b), a 6D teach mouse input device, and some sensors. Various data can be recorded during a demonstration for further processing in the PbD system running on a workstation. The objective is to explore the possibilities of integrating learning and clustering algorithms for automated robot programming. In particular it is investigated how human interaction within system- as well as user-initiated dialogs can support the induction component.

Keywords: Programming by Demonstration (PbD), Robot Programming by Demonstration (RPD), Teaching, Robots, Manipulators, User Intentions, Dialog-Based Learning (DBL)

1. Introduction

Programming by Demonstration (PbD) is an approach for faster software development in domains where the user works interactively with a computer or some machinery. Such domains are the programming of graphical user interfaces as well as graphics and text editors where the user interacts with the editor program by using keyboard, mouse, pen, or other input devices. Another important domain is the programming of robots, appliances, and other machines where the user moves and manipulates the device by provided control panels. In all cases the goal of PbD is to “show” the system a new functionality by demonstrating the desired behavior, rather than coding it in a programming language. What has been shown is recorded as traces of sensor data, transformed into basic actions, analyzed, and then a computer program is generated that allows the user to repeat the demonstrated task

by calling one command. To make the program even more useful, additional generalization methods have to be applied during the generation of the function.

Imagine the problem of assembling something with a robot manipulator. To program such a task today, often so-called teach-in methods are used. A “programmer” directs the robot’s end-effector to all necessary locations, where it has to grip or manipulate something, by means of six buttons on a teach panel (three buttons for x-, y-, and z-direction, three buttons for the respective orientations). At each important position the teacher stores the location as a 6-tuple, and additional commands like open or close gripper. Replaying the whole sequence of stored “button hits” repeats exactly the taught program. Teach-in programming of manipulators can take a lot of time and is really acceptable only if large numbers of exactly the same product items shall be assembled.

Programming by demonstration with some generalization capabilities overcomes the un-economic usage of teach-in especially for the manufacturing of a product with many possible variants. Let us take as a most advanced scenario the assembly of a torch. Wouldn’t it be comfortable to just assemble it with one’s own hands, watched by video and laser cameras and maybe wearing data gloves, i. e., sensors that provide the data to automatically generate the necessary robot program for the assembly task? And wouldn’t it be even more comfortable to demonstrate the task with few different torches, but achieving an assembly function for all possible variants of them?

Unfortunately, PbD settings as described in this scenario are not yet available neither in industry nor at universities or are only partially realized. The reasons are manifold: Sensors are still too expensive, and the sensor processing of complex image sequences is not fast and accurate enough to fit the strong needs of many manufacturing processes. A human arm and hand with their more than 30 degrees of freedom (DOF) are more flexible in performing object manipulations than an ordinary 6-DOF robot with its gripper. Another problem is that one single human is mostly not sufficient to demonstrate complex production or assembly sequences. There is always a variety of interactions between machines, robots, and humans. In addition, not all steps of an assembly are usually performed at one place. The whole process is a sophisticated sequence of time-sliced actions at different stations along a conveyor belt. Therefore, programming the whole torch assembly task by demonstration can probably not be done as sketched in the scenario.

However, there are industrial “subtasks” that could already be programmed by demonstration today—even with generalization—and would significantly reduce the required programming effort. Such subtasks are pick&place macros for different settings, the loading of pallets, sorting tasks, part feeding tasks, etc. They are far from complicated assembly tasks, but still need the same time-consuming and awkward teach-in programming procedure. (Instead, one could also use graphical off line programming, but that method also needs a significant amount of teach-in postprocessing in order to adjust the generated program to the real setting (Neubauer et al., 1992).)

Robot subtasks like pick&place or sorting objects often contain the need for the generalization of geometrical positions, of constants to variables, and of similar objects to classes. There is also a need for finding loops or recursive calling schemata in the action traces. Such generalizations are well-known from inductive machine learning, in particular explored under the label of *learning from examples*. In fact, as in most practical applications

(see the discussion in (Thrun & Mitchell, 1993)), a combination of analytical and inductive learning is used for robot programming by demonstration (RPD). This means that previous knowledge of the domain and particular knowledge about tasks and environments have to be available.

Nevertheless, PbD and RPD have to cope with inherent problems that do not necessarily exist for all learning-from-examples tasks. There are only *very few* examples available, typically one to five, because nobody wants to demonstrate the same task very often. This causes great uncertainty about what the user intended with the demonstrated examples, particular because the hypothesis spaces are large. As a way to overcome this problem we propose dialog-based learning (DBL) (Bocionek & Sassin, 1993), a means to engage the demonstrating person in the induction process by question&answer dialogs that may be initiated by the RPD system as well as by the person. The usefulness of DBL has already been shown in other PbD domains in which—based on only few examples—functions for graphics editors (Sassin, 1994) and personal assistance software (Bocionek, 1995) could be learned (induced).

The remainder of this article is organized as follows. In section 2 we will describe the principle of robot programming by demonstration in more detail and show our solution for an adequate system architecture. Section 3 contains the description of our RPD setting and describes in detail the two major phases of RPD. In section 4 we will give a detailed example where human interaction is used during the induction process. A discussion of the work related to our approach follows in section 5. Finally, section 6 concludes the paper and outlines planned future work.

2. Robot programming by demonstration

The key characteristic of robots is versatility; (...). (...), but it can be exploited only if the robot can be programmed easily. In some cases, the lack of adequate programming tools can make some tasks impossible to perform. In other cases, the cost of programming may be a significant fraction of the total cost of an application. For these reasons, robot programming systems play a crucial role in robot development.

(Tomás Lozano-Pérez, 1983)

Having in mind Lozano-Pérez' demand for an adequate robot programming environment, we want to give a short overview on common programming methods and describe the principles of our approach, called Robot Programming by Demonstration (RPD).

2.1. Traditional robot programming methods

Today, many different programming methods with a large number of variants exist. In (Lozano-Pérez, 1983), three major categories can be identified:

1. guiding;
2. explicit or robot-level programming;

3. implicit or task-level programming.

In *guiding* (also known as *teaching by showing*, *teach-in*, or *play-back*), the operator moves the manipulator and records explicitly all relevant positions. The major advantages of this method are that only positions are taught that can really be reached by the robot and that the explicit recording leads to very precise trajectories. Furthermore, this method is very easy to perform. However, it has some major drawbacks: The robot cannot be used for the production process during the teaching phase and the taught sequence cannot be altered; it is usually even impossible to include sensor signals in the generated program. Additionally, only the robot used for the teaching can exactly follow the recorded path¹.

Explicit or off line programming in a programming language introduces control structures like loops and branches, and offers the chance to react to external events by using sensors, thus leading to very flexible robot programs. For the programming process, either special-purpose languages like VAL II (Versatile Assembly Language) or general-purpose languages with special extensions, like PasRo (**P**ascal for **R**obots), are used. These languages offer much more flexibility than teach-in, and the robot can be programmed off line, i. e. the production process does not have to be interrupted. But, due to the richness and power of these languages, programs are hard to write by robot operators who often perform the teach-in process today. Therefore, usually only highly qualified persons can cope with this method.

The most advanced of the three above-mentioned methods for the programming of robots is *implicit programming*. This method requires a sophisticated planning system, because no paths or positions have to be specified, but the programmer only determines the goals and subgoals of a task which are then further detailed by the planner until the robot's execution level is reached. Problems with this method arise due to the high complexity of the tasks which require planning systems that are not available for industrial usage today.

To complicate the situation, all three methods can be extended and combined. For example, in spot or arc welding the data of the CAD model is used and adjusted with teach-in methods to generate the robot's path. To simplify the programming, graphical simulation systems are used in combination with other methods. This *off line programming* offers the chance to test several alternatives and to visualize the simulated execution. As an extension of graphical simulation, virtual environments (VE) are explored for robot programming, too (see, e. g. (Flaig et al., 1994)).

With robot programming by demonstration (RPD), we want to develop a method that offers richness, power, and flexibility, but is as easy to use as simple teach-in. The principles of RPD are described in the next section.

2.2. Principles of RPD

Robot programming by demonstration is an intuitive method that makes the programming of robots easier and faster for experts, but which can also be used by robot operators due to its simplicity. The idea of RPD is to combine teach-in with learning and to integrate the user in all processing and induction steps.

Learning in an RPD system can be realized at different levels of abstraction. Therefore, skill as well as knowledge acquisition by the system via human demonstration might be integrated in the system (Friedrich & Kaiser, 1995). The different levels are shown in fig. 1.

Figure 1. The principle architecture for an integrated RPD system. The learning components are emphasized in the sketch, showing the major input data for the different modules.

The two major levels of an integrated RPD system are as follows:

1. *Learning on the symbolic level:* The system can either generalize robot programs completely or partly, using sequences of elementary operations as input data for the learning process (Münch et al., 1994; 1994b); or the initial acquisition, extension, and refinement of robot action knowledge and object descriptions may be realized by processing data extracted from experiments (Kreuziger & Hauser, 1993).
2. *Learning on the subsymbolic level:* On the lower level, either trajectories can be reconstructed from freehand motions (Ude, 1993; Ude et al., 1994) or robot controllers performing basic skills might be generated with the help of artificial neural networks (Kaiser et al., 1994).

In the system described within this article, we will mainly concentrate on learning robot programs on the symbolic level.

The system architecture developed for the realization of the RPD prototype system is shown in fig. 2. The user interface on the left consists of the 6D mouse for moving the

robot, and of some windows on a workstation that enable the user to guide the induction by supervising the generalizations proposed by the system and supplying her or his intentions.

Figure 2. Architecture of the realized RPD prototype system: The interfaces to the user (left) and the robot system (right) are enclosing the RPD kernel system.

The system's kernel is located in the center of the figure. The *analyzer* component performs the required preprocessing steps on the recorded demonstration trace. This preprocessing consists mainly of segmenting the trace and classifying the segments with respect to the given *planning language*. This planning language is basically structured like a conventional programming language, consisting of branches, loops, and *Basic Operations (BOs)*. However, by applying the analyzer only BOs are directly generated from the trace. All control and hierarchical structures are introduced by the *induction component* which operates on the planning language only. The induction of branches and variables is supported by the user. The result of the induction is stored in a knowledge base for later execution as a single plan, but also for use as a planning macro by the execution unit.

The right part of fig. 2 shows the robot's environment for executing the generated program. The program is in a kind of "neutral" format at this point, therefore it has to be transformed into one of several robot programming or simulation languages in order to execute it either with a real robot or with a simulation system.

Three major phases can be identified in the RPD system developed so far: Firstly, the *demonstration phase* in which a solution for the task is demonstrated by the user. All data of this demonstration is stored in a trace for further processing. Thereafter the analyzer component is applied to transform the stored trace on the symbolic level, namely to the representation given by the planning language used in the succeeding components of the system. Secondly, the *induction phase* takes place. Therein the output of the analyzer is

taken and processed with respect to additional information—the user’s intentions acquired through a man-machine interface. Finally, the *execution phase* is performed in case a previously generated program shall be applied. This includes selecting, sequentializing, and parametrizing the program as well as driving the selected robot. All three phases will be described in more detail in the following section, whereas an example for the whole process is shown in section 4.

3. The RPD environment

After the principles of RPD have been outlined, this section describes the function and the architecture of the RPD system which has been realized prototypically at the Institute for Real-Time Computer Systems and Robotics, University of Karlsruhe (see fig. 3). The techniques and methods underlying the three already mentioned processing phases are described in detail. Preceding this description, a short overview of the planning language used by the system’s modules is given.

Figure 3. A sample RPD testbed with a PUMA 260b robot and a 6D bowl, realized at the Institute for Real-Time Computer Systems and Robotics, University of Karlsruhe.

The Planning Language

The language providing the symbolic representation used for generating and processing programs from demonstrations basically consists of five *Basic Operations (BOs)* that are:

`grip(O)`, `ungrip(O)`, `approach(s, d)`, `depart(s, d)`, and `transfer(s, d)`.

Here, *O* denotes either an object of the world model or a relational condition that can be evaluated and unified with the world model in order to determine an object. *s* and *d* are frames (also called *locations*) describing positions and orientations of the manipulators tool center point.

Like most procedural programming languages, the planning language includes higher level elements that represent branches and loops. These are *if-then-else* branches as well as *while* and *repeat* loops. Their body holds a sequence of other operators that can be branches, loops, or even whole plans themselves. A *plan* is an operator containing a sequence of operators and representing the solution to a given task. It can itself be used as part of another plan, branch, or loop. Currently, the loop operators are not used. However, they are already defined for the next development stage of the system.

All operators of the planning language are modeled in the tradition of STRIPS operators (see Fikes & Nilsson, 1971), i. e. for every operator its applicability depends on the evaluation of a *precondition* and after the application a certain *postcondition* holds.

Conditions themselves are expressed in terms of combinations of disjunctions and conjunctions of relations. These are describing relationships between objects like *on*(O1, O2) and *in*(O1, O2) or object-features like *obj-type*(O, type). Terms are always given and transformed into *disjunctive normal form (DNF)*. They can be evaluated with respect to the world model that is given initially and updated each time one of the operators described above is applied. The implemented variable concept allows to use variables as attributes of relations. Unification of a term holding variable attributes allows to determine all combinations of variable instantiations on the current world model that yield a positive evaluation of the term.

3.1. Demonstration Phase

Demonstrating a task

In order to generate a *new* robot program, the PbD process starts with the *demonstration of an example*. Several different methods exist for that step, each with particular strengths and drawbacks which will not be discussed here in detail. In the system presented here, a 6D mouse is used to control the PUMA 260b robot (see fig. 3), because only a short training period is needed to generate good demonstration data.

Filtering

Pauses occurring in demonstrations and unsteady motions due to not perfectly trained users result in noisy data². Therefore, a *filtering* of the recorded trace follows the demonstration in order to remove redundant vectors to reduce the amount of data (see fig. 7).

Analysis

The next step is the *analysis* of the filtered trace, which is mainly used to transform the “raw data” into the symbolic format of the planning language. To achieve a correct transformation, i. e. the recognition of the correct sequence of BOs hidden in the recorded vector sequence, two problems have to be solved. Firstly the trace has to be segmented into subsequences, each of them representing a certain BO. Secondly each segment has to be

classified and mapped on one of the five BOs. Obviously the two tasks of *segmentation* and *classification* depend on each other and therefore have to be done simultaneously. In this sense the transformation task is similar to the problem of continuous speech recognition. Another aspect also common to the speech recognition domain is that of time variations. Just like people speak with different speeds, various users perform demonstration movements with different speeds. Moreover, the duration of a movement usually depends on the distance between the start and the final location.

In the domain of speech recognition *Time Delay Neural Networks (TDNNs)* have been proven to be powerful tools for various segmentation and classification problems (Waibel et al., 1989), overcoming also the problem of time variations in the input signal. For this reason the application of TDNNs was considered for the solution of the analysis problem in the RPD system. For each of the BOs a TDNN was generated, consisting of three layers with 5 neurons in the input and exactly one in the output layer. The number of neurons in the hidden layer varies for the different BOs as well as the applied time-delays and training epochs, as can be seen from table 1.

Table 1. Some parameters of the TDNNs for Basic Operation recognition

BO Net	Perceptrons in the hidden-layer	Time-delays on 1st layer	Number of training epochs
transfer	9	10	10,000
approach	10	15	12,000
depart	10	15	12,000
grip	8	8	10,000
ungrip	8	8	10,000

Motions are restricted to translational movements and each net was trained with nine to ten training sequences (10–20 vectors per training sequence) of pick&place operations. The demonstrations vector sequence is fed simultaneously into each BO TDNN. By studying the outputs produced, i. e. the net activations, a decision about which BO was performed during what time-span of the demonstration is taken.

Automatic correction of classification errors, like a missing gripper event between a detected approach and depart pair is done by applying some heuristics about possible operation sequences in a post-processing step. Following this stage the transformation from the subsymbolic to the symbolic level takes place. The determined sequence of Basic Operations of the planning language is generated and instantiated with respect to the segmented vector sequence.

For manual correction of classification errors and supervision of the analysis module's results, a graphical user interface is provided (see fig. 9). The manipulator's trajectory is plotted divided in its x, y and z coordinates along the time axis. Moreover, the boundaries of the recognized BOs are given. Now the user has the possibility to correct the BO sequence in order to improve the classification results. Collected corrections from multiple demonstrations might be used later to retrain the TDNNs and thus improve the automatic analysis performance of the system over time.

3.2. Induction Phase

Constructing a plan from the BO sequence generated by the *analysis* module is a task consisting of the following phases that are all performed by the *induction* unit:

1. Formation of *Object Groups (OGs)*, where each OG contains a subset of the BO sequence that represents the manipulation of exactly one object.
2. Determination of the valid postcondition of each BO sequence of step 1.
3. Acquisition of the *user's intention* regarding the demonstration via man-machine communication.
4. Detection and generation of branches, i. e. conditioned manipulations and elimination of superfluous manipulations.
5. Acquiring the *object selection criteria* for the manipulated objects and introducing variables based on user interaction.

These steps will be discussed in more detail in the following paragraphs.

Preparing BOs; Grouping & Calculating Conditions

For further processing of the given BO sequence, it is structured into subsets, the already mentioned Object Groups³. At this stage of the system's development, there is no interaction of the manipulated object with other objects like, e. g., insertion operations. Therefore, division and grouping of the BO sequence into a sequence of OGs is fairly simple. The graph in fig. 4 represents the possible sequences of BOs with and without interactions for a one arm robot. Following this graph OGs are built in a simple straightforward way.

Starting from the *transfer* towards an object, proceeding with its *grip*, the sequence of Basic Operations is traced until the object is finally *ungripped* and the manipulator *departed* from it. Therefore, each Object Group encloses a subpart of the users demonstration that is characterized by the manipulation of one specific object.

As can be seen in fig. 4, the consideration of interactions of objects enables multiple cycles of *transfers* to other objects and interactions of the *gripped* object with those. These interactions take place in between the *grip* and *ungrip* of the primarily manipulated object.

After the identification of all Object Groups, a forward trace over the BO sequence is performed. In this process the fully parametrized pre- and postconditions of every BO are determined. Also the postconditions of every Object Group and finally the whole BO sequence, i. e. the demonstration itself, are determined. Every type of BO has certain preconditions and postconditions that have to be valid for its application respectively are valid after application. In addition, there are pre- and postconditions that depend on the environmental context in which the BO is executed. If, for example, the postcondition for the BO *approach(s, d)* has to be determined, the result depends on whether an object has already been grasped or not. If the gripper does not hold an object (i. e., it is free), *approach*

Figure 4. Graph representation of possible BO sequences

is prior to a gripping action. Therefore, the gripper encloses the object o to be gripped at the end of the approach. This forces the insertion of the $\text{enclosed}(O, \text{Manipulator})$ relation to the postcondition of the approach. A special case is given when an object is ungripped. Here, all possible relations to all other objects have to be tested and added to the ungrip BO's postcondition.

While forward tracing through the BO sequence and determining the conditions of the Basic Operators, the world model is updated by virtually applying the BOs to it. Starting with the first BO and the initial environmental configuration, the world model data is updated sequentially after the determination of each operators conditions with respect to the operators definition.

As soon as the conditions of all BOs related to an OG are determined, they can be fused to the postcondition of the group. Naturally this includes to check for relations built up by one BO that were destroyed later by another BO related to the same Object Group. The same fusing of postconditions is done for all OGs in order to determine the whole demonstration's postcondition. In other words, the demonstration's postcondition is the accumulation of all lasting changes that occurred throughout the demonstration.

But do all these changes really reflect exactly the users intention? Or were some actions simply forced by the environmental conditions present at that moment? Or were some of the manipulations even unnecessary in order to solve the given task?

Acquiring new Knowledge by User Interaction

Unfortunately a computer system is not able to *guess* the answers to the questions raised above. Since the primary focus in the proposed PbD system is not to induce generalizations and/or knowledge from a large number of examples, a different direction than classical ML

induction techniques has to be followed to generate the desired answers. In order to keep the number of required demonstrations as small as possible another source of information has to be integrated in the induction process—the user itself.

The postconditions of the demonstration that are related to the same Object Group are presented to the user successively. Thus he can select the conditions that reflect his intention best and can discard the others. Thereby the system gathers the information necessary to identify actions which are conditioned with respect to the user's intention. With this knowledge, branches can be derived and *if-then-else* operators can be generated. Parts of the demonstration that were found to be superfluous can even be deleted.

Generalizing and Optimizing; Branching & Garbage Collection

While the processing strategy in the first phase of the induction process was characterized by forward tracing through the input BO sequence, the sequence is now traced in the reverse direction, i. e. backwards. Tracing backward, branching, determination of preconditions of the Object Groups, and optimization by deletion of superfluous manipulations is done.

In the following, the developed branching algorithm is given in pseudo-code in which the initializations and update rules are given precisely. *I* denotes the users intention, *UI* unsatisfied intentions, *UPrec* unsatisfied preconditions, *PostC*(*og_i*) the postcondition and *PreC*(*og_i*) the precondition of argument *og_i*.

```

1.  determine-branches (plan, I):
2.      PostC(plan) := UI := I;
3.      UPreC := ∅;
4.      loop OGcurrent := OGlast to OGfirst                                /* backward trace */
5.          determine-precondition (OGcurrent);
6.          if direct-contribution (OGcurrent, UI) then
7.              UI := (UI / PostC(OGcurrent));                                /* intended manipulation */
8.              UPreC := (UPrec / PostC(OGcurrent)) ∧ Prec(OGcurrent);
9.          elseif indirect-contribution (OGcurrent, UPreC) then ;
10.             generate-branch(OGcurrent);                                /* conditioned manipulation */
11.             (body branch) := OGcurrent;                                /* generate branch */
12.             BC := (equal (PostC(OGcurrent) ∩ UPreC) nil);
13.             UPreC := (UPrec ∨ ((UPrec / PostC(OGcurrent)) ∧ Prec(OGcurrent)));
14.         else                                                                /* superfluous manipulation */
15.             dump (OGcurrent);
16.     end-loop;
17.     Prec(plan) := UPreC;
18.     return (plan);
19. end determine-branches.
```

For each Object Group checked during the backward trace three possible reasons for its occurrence exist:

1. The OG was *intended*. This means to perform the related manipulation is crucial for the achievement of the intended goal of the demonstration, namely the previously acquired user intention.
2. The OG was *not intended* but is nevertheless crucial for the achievement of the desired goal, since it enables an intended OG following later on.
3. The OG is *superfluous* and can be omitted without thwarting the achievement of the desired goal configuration.

OGs of the first kind are called to be *directly contributing* to the demonstrations, i. e. the user's goal. To test whether an Object Group is directly contributing its postcondition is simply checked against the set of conditions that represent the *unsatisfied intentions* (*UI*). If there is an intersection the OG creates parts of the user intended goal state—it is directly contributing.

The second type of OGs is called to be *indirectly contributing* to the demonstration. Here, the postcondition has an empty intersection set with the unsatisfied user intentions, but there is an intersection with the set representing the *unsatisfied preconditions* (*UPreC*) of the following OGs. The term “intersection” used in the last phrase is meant in the way that parts of the OGs postcondition enable the positive evaluation of parts of the *UPreCs*. For example, we assume that as a result of a manipulation free space is provided to be occupied by another object. This might enable a manipulation that requires a subpart of this space for delivering some object. Here the amount and location of newly provided space might not be equivalent to the required space but nevertheless there is a dependency; an intersection.

The last type of OGs is fairly easy to handle. It simply does not contribute to the intended goal at all, neither directly nor indirectly.

Now the three different types of Object Groups are defined and it has been shown how a particular OG can be classified as one of these types. The remaining question is: how is an OG of a particular type to be treated during the performed backward trace in order to optimize or generalize the plan?

As outlined in the algorithm given above, a directly contributing OG is kept without any changes. Its contribution is deleted from the unsatisfied intentions and its precondition is added to the unsatisfied preconditions. Furthermore, possible indirect contributions are deleted from the *UPreCs*. In the contrary, OGs that are not contributing at all are simply deleted. This is in fact an optimization since an unnecessary motion sequence is erased. The interesting OGs in terms of generalization are indirectly contributing ones. Each of those is embedded in a newly generated branch. The *branching condition* (*BC*) is instantiated with an OGs indirect contribution, namely the intersection of the OGs postcondition with the unsatisfied preconditions. The unsatisfied intentions remain unchanged because there is no direct contribution. However, the unsatisfied preconditions are updated by a disjunction. One disjunctive term being the unchanged former *UPreCs*, corresponding to the case when the branch is not carried out. The other disjunctive term being the former *UPreCs* with the OGs indirect contribution deleted and the precondition of the branches OG added. This corresponds to the case in which the manipulations embedded in the branch are carried out.

Backtracing through the whole demonstration, generating branches when possible, and updating the unsatisfied preconditions leads to a generalized and optimized plan. Its precondition which determines its applicability is equivalent to the set of accumulated unsatisfied preconditions. Obviously the “factor” of generalization corresponds directly to the number of disjunctive terms in the precondition of the resulting plan, which depends on the number of conditioned manipulations that were detected throughout the backward trace.

Further Generalization; Object Selection Criteria & Variables

Two methods exist for further user guided generalizations. The first one consists of editing the selection criteria that are determining which objects are chosen for manipulation when applying a plan. Initially an *object selection criterion (OSC)* is given from the analyzer module for every *grip* operation and therewith for every OG. This initial OSC is a conjunction of the gripping width and the position the manipulator’s *Tool Center Point* had at the time the *grip* took place. Based on this initial OSC and the world model, the induction module determines the manipulated object. As soon as the object is determined the OSC is expanded with all relations of the object held at the time it was gripped. This includes the objects type (e. g. *cube*, *cylinder*, ...) as well as all valid inter object relations (e. g. *on*, *in*, ...). Thereafter the mostly constrained OSC possible is given for every OG. Based on user interaction the constraints are smoothed by introducing variables and deleting parts (literals) of the OSC. Backpropagating these changes to the plans precondition results in a generalization, widening the applicability of the plan.

Moreover, variables can be introduced by user interaction in the postcondition of the plan which reflects the users intention. For example, we assume that an object O was put into another one (O_c) which is described as being of type *case*. Furthermore these facts shall be represented in the user’s intention as the relations $\text{in}(O, O_c)$ and $\text{type}(O_c, \text{case})$. Introducing a variable for the type *case* and backpropagating this change throughout the whole plan widens its applicability since every object able to serve as a container for O can be used to instantiate O_c , not only one of type *case*.

Finally, the induction component stores the generated plan in a knowledge base. In the storing process all objects are replaced by variables. This enables the unification process necessary for plan selection and instantiation as described in section 3.3 below. Thus, generated plans can be applied to given tasks by the system’s execution components. The representation of whole plans as STRIPS operators having pre- and postcondition also allows to use them as macro operators in a planning process. This way more complex tasks can be solved by combining plans that were generated from different demonstrations. All steps described so far are shown schematically in the left part of fig. 5.

3.3. Execution Phase

The third processing phase (depicted in the right part of fig. 5) addresses the problem how to select an applicable plan to a given task among previously generated ones, how to instantiate

Figure 5. Phase 1 (demonstration, analysis, and induction) and Phase 2 (completion and execution) of the RPD process. Dashed arrows indicate the integration of the user.

its variables with respect to the current world model, and finally how to execute it with a given manipulator.

Plan Selection

For the selection of a plan that enables the robot system to solve a given task two prerequisites have to be fulfilled: (i) the task's goal specification has to be given as a combination of the predefined relations in *disjunctive normal form (DNF)*; (ii) the current environmental configuration has to be given as a valid world model. Given these two inputs a plan is selected by firstly matching the tasks goal specification with the postconditions of all available plans. All plans that are left after this process serve as candidates for the second selection phase. In this phase, each plan's precondition is matched with the given world model. Plans that "survive" this phase are applicable and do lead to the desired goal. Currently the first plan found to fulfill both selection phases is chosen for execution.

However, this is a greedy method leading mostly to a suboptimal selection. Future extensions will be added to assess applicable appropriate plans in terms of execution time, complexity of required motions etc. to enable a better selection.

Preparing the Execution; Plan Instantiation and Sequencing

At first glance the problem of how to prepare the selected plan for execution, namely how to instantiate its free variables and how to select the appropriate objects in the current world configuration, seems to be difficult to solve. Surprisingly, to determine the instantiations is not difficult at all. In fact, these instantiations are provided as a side effect of the selection phase. The matching processes required for plan selection include unifications of the objects in the world model with the variables in the plan's conditions, followed by the evaluation of these instantiated conditions on the world model. Therefore the determination of an applicable and appropriate plan includes one or more sets of possible and valid instantiations from which a random one is chosen. As in the selection process, the optimization of the selection of a valid instantiation set will be a future system's extension.

Based on the world model, the instantiated plan is transformed into a sequence of Basic Operations, i. e. branching conditions are tested and the bodies of the corresponding branches are either omitted or prepared for execution.

Solving the Task; Plan Execution

The last step is the execution of the generated program. Usually, the same robot that was used for the demonstration will be used for the execution, too. But, due to the generality of the program, it is also possible to execute the instantiated plan with another robot or with a simulation system (see fig. 5, right). Of course, the basic capabilities (degrees of freedom, sensory equipment, work space) of the system used for the execution must not differ from those supported by the system used for the demonstration. In our system, the generated program that consists of a set of basic operations can either be transformed into VAL II commands (the programming language of the controller used with our PUMA 260b) or into commands suitable for a graphical simulation interface. A transformation into other robot programming or simulation system languages is no principle problem, since only the module performing the transformation from the planning BOs to the desired language has to be added.

4. Interaction with the user—an example

In this section, the process of programming a task by demonstration, employing the methods described in section 3, is illustrated by following an example through the phases of *demonstration* and *induction*. The example is demonstrated and the generated program executed with a real industrial robot, a PUMA 260b. Thereby the system overcomes one of the most criticized aspects of former *Robot PbD systems* which have solved problems in simulated environments only (Segre, 1988) or have treated the robot as a dimension-less point in space (Andrea, 1985).

Setting up the Scene; The Example

The example task chosen for illustrating the methods used in the developed PbD system is shown in fig. 6. Two cylinders shall be placed in two designated compartments of a case.

Figure 6. The start (left) and end (right) configuration of the experiment described in this section. The numbers on the arrows indicate the order in which the manipulations were performed.

As indicated by the arrows in fig. 6, the user performs four manipulations while demonstrating the task. Two of those are required in any case, namely the manipulations of the cylinders (arrows 1 and 4). The manipulation indicated by arrow 2 is required due to the given start configuration but would be obsolete if the compartment was empty from the beginning. Therefore, a branch has to be detected and generated during the induction process. The 3rd manipulation is concerned with moving the second cube needlessly with respect to the task. This motion sequence has to be eliminated later on during the induction process.

Demonstration Phase

The data recorded during the demonstration are the position of the tool center point (TCP) of the robot, the distance between the gripper jaws, and the values of the 6D bowl moved by the user. More data available like forces, torques, and orientations are not taken into account in this example. A snapshot of the raw trace recorded during the demonstration

((0 0 0 0))	(219.86 222.71 428.43)	(4939))	
((0 0 -5 0))	(219.76 222.55 429.18)	(4939))	
((0 0 0 0))	(219.76 222.55 429.18)	(4939))	
((0 0 0 2))	(219.74 222.51 429.18)	(2692))	$\Rightarrow 0 0 0 2 0$
((0 0 0 0))	(219.74 222.51 429.18)	(2692))	
((0 0 40 0))	(219.79 222.74 424.78)	(2692))	$\Rightarrow 0 0 4 0 4$
((0 0 42 0))	(219.96 223.07 394.19)	(2692))	$\Rightarrow 0 0 31 0 31$
((-1 0 0 0))	(219.86 253.47 375.73)	(2692))	$\Rightarrow 0 30 18 0 35$
((0 0 37 0))	(219.15 265.87 362.32)	(2692))	$\Rightarrow 1 13 14 0 19$
((0 0 37 0))	(219.23 265.68 336.63)	(2692))	$\Rightarrow 0 0 25 0 25$
((0 0 0 0))	(219.34 266.01 320.16)	(2692))	$\Rightarrow 0 0 17 0 17$
((0 0 0 0))	(219.36 266.08 320.13)	(2692))	
((0 0 0 0))	(219.37 266.11 320.10)	(2692))	

Figure 7. Recorded trace data containing raw unfiltered data. Left part (from left to right): position & gripper values of the 6D bowl, position of the TCP, and gripper width. Right part: $\Delta(x_t, x_{t-1})$, gripper event & distance(x_t, x_{t-1}).

and its filtered representation is shown in fig. 7. The sequence shows a `grip` and part of a `depart` operation.

As can be seen from the data in fig. 7 (left side), sometimes the operator does not move the robot’s arm at all. These standstills or “zero-movements” are eliminated by the filtering process whose parameters can be modified by the user if desired.

After the trace has been filtered, the TDNNs are applied to segment the trace and classify the segments into the five classes of *Basic Operations*. Part of the result of this step is shown in fig. 8.

(0 2)	$\Rightarrow$ transfer ((220, 92, 378), (220, 223, 402))
(1 14)	$\Rightarrow$ approach((220, 223, 402), (220, 223, 429))
(3 17)	$\Rightarrow$ grip ((220, 223, 429), (2692))
(2 21)	$\Rightarrow$ depart ((220, 223, 425), (202, 255, 320))
(0 2)	$\Rightarrow$ transfer ((202, 255, 320), (-69, 270, 334))

Figure 8. Analysis results employing TDNNs; from left to right: BO identifier, start vector in filtered trace, and symbolic BO representation. `grip` & `depart` correspond to the data in fig. 7.

After the analysis the trace is presented to the user by a graphical tool. This allows to check the results of the previously performed automatic segmentation and classification. Figure 9 shows a part of the example trace. For illustrating the accuracy of the TDNN based classification, two segments that were classified to represent `approach` BOs were highlighted. The high classification accuracy can be evaluated by comparing the `approach` segments with the lightest curve which represents the Δz -values of the trace. This shows the high performance of the automatic analysis since an `approach` operation in the experiment is mainly characterized by negative translational Δz -values of the TCP.

The *demonstration* phase ends with the storage of the determined BO sequence (fig. 8).

Figure 9. A snapshot of the example trace, showing translational Δx , Δy , and Δz (darkest curve) TCP values, gripper events, and two approach segments (grey).

Induction Phase

As described in section 3, Object Groups are formed by the induction module and the demonstrations postcondition is determined (see fig. 10).

```
(position(cyl1,(-68 240 530)) ∧ in(cyl1,shelf9) ∧ type(cyl1,cylinder) ∧ space(cylinder,(220 225 460)) ∧
position(cub1,(225 205 460)) ∧ on(cub1,tray) ∧ type(cub1,cube) ∧ position(cub2,(125 245 460)) ∧
on(cub2,tray) ∧ type(cub2,cube) ∧ space(cube,(-160 130 530)) ∧ position(cyl2,(-70 180 530)) ∧
in(cyl1,shelf10) ∧ type(cyl1,cylinder) ∧ space(cylinder,(307 65 460)))
```

Figure 10. The determined postcondition of the demonstration.

In the next step, the user has to select the intended subparts of the postcondition determined by the system. In our example, the manipulations of the two cylinders were intended which is reflected by the specified user intention shown in fig. 11.

```
(position(cyl1,(-68 240 530)) ∧ in(cyl1,shelf9) ∧ type(cyl1,cylinder) ∧ position(cyl2,(-70 180 530)) ∧
in(cyl2,shelf10) ∧ type(cyl2,cylinder))
```

Figure 11. The subpart of the plan's postcondition reflecting the intention of the user.

Applying the generalization procedure presented in section 3 leads to the generalized plan shown in fig. 12. The superfluous manipulation of the cube has been eliminated and one branch has been generated reflecting the manipulation of the other cube, which was contributing only indirectly to the solution of the example task.

By specifying the initial positions of the two cylinders on the tray as variables, further generalization is achieved. Thus the precondition of the final plan shown in fig. 13 determines the plan to be applicable when there are two cylinders on the tray, there is space in

```

OG1                ;; put first cylinder in the shelf
If (equal (space(cylinder, (-70 180 530))) nil)
  OG2                ;; manipulate cube if necessary
OG4                ;; put second cylinder in the shelf

```

Figure 12. The generalized plan (manipulations are abbreviated by their Object Groups).

the first target shelf, and either also space in the second target shelf or a cube to be removed from it.

$$\begin{aligned}
 & ((\text{on}(\text{cyl1}, \text{tray}) \wedge \text{type}(\text{cyl1}, \text{cylinder}) \wedge \text{space}(\text{cylinder}, (-68\ 240\ 530)) \wedge \text{on}(\text{cyl2}, \text{tray}) \wedge \text{type}(\text{cyl2}, \text{cylinder}) \wedge \\
 & \quad \text{space}(\text{cylinder}, (-70\ 180\ 530))) \\
 & \quad \vee \\
 & (\text{on}(\text{cyl1}, \text{tray}) \wedge \text{type}(\text{cyl1}, \text{cylinder}) \wedge \text{space}(\text{cylinder}, (-68\ 240\ 530)) \wedge \text{on}(\text{cyl2}, \text{tray}) \wedge \text{type}(\text{cyl2}, \text{cylinder}) \wedge \\
 & \quad \text{in}(\text{cub1}, \text{shelf10}) \wedge \text{type}(\text{cub1}, \text{cube}) \wedge \text{position}(\text{cub1}, (-72\ 170\ 530)) \wedge \text{space}(\text{cub1}, (225\ 205\ 460))))
 \end{aligned}$$

Figure 13. The precondition of the generated generalized program, determining its applicability.

5. Related work

In this section, the relationship of RPD to other research work in this area will be discussed. Due to the number of existing approaches, it is neither possible to discuss every system nor to describe each selected approach in detail. Since the RPD approach presented here is more focussed on the acquisition of procedural knowledge, the work which is more related to adaptive control method as e. g. in (Liu & Asada, 1993) or (Atkeson et al., 1987) will not be reviewed. In the following, two different “classes” of RPD will be distinguished: RPD with direct repetition and RPD with generalized repetition.

5.1. RPD with direct repetition

The term *RPD with direct repetition* is expressing that the demonstrated sequence of robot actions is used to exactly repeat the same task. The generated program cannot be applied to even slightly modified products, environments, or manufacturing situations (see section 1 above and section 12.3 in (McKerrow, 1991)). In particular, RPD with direct repetition is interesting for manufacturers in order to shorten programming cycles, provided that humans could demonstrate a task simply with their hands, i. e. in the most “natural way”, while video and/or laser range cameras watch the scene. Two research projects explore exactly this technique.

Kuniyoshi et al. (1994) have called their RPD approach with direct repetition *Teaching by Showing*. A human worker builds various structures with different blocks using her or his hands while being observed by a stereo vision system. With several interpretation modules

(like the movement detector, the hand tracker, the object finder, the edge junction classifier), all information necessary for the action recognizer are extracted from the image sequences. The recognizer generates a script-like action model that symbolically describes the extracted motion sequence. The recognition is model-based, i. e., it uses predefined action templates for large transfer motions as well as local motions like *approach* and *depart* movements. The recognizer instantiates those templates with the names of the identified objects, their coordinates, and spatial relation information (like alignments between objects). Finally, LISP operators like *place-on-table*, *pick* or *reach-with-obj* are created which can be sent to the robot control in order to repeat the task.

Since the RPD approach presented within this article does not use a vision system but only sequences of position tuples coming from the 6D mouse's interface, the system cannot profit from the data that an object recognizer delivers, or from spatial relations between objects detected by the edge junction classifier. Therefore, more information has to be provided to it beforehand, especially the initial position of the objects in the scene. Furthermore, the dialog-based approach for segmenting the flat sequence of positions is partly equivalent to the functionality of Kuniyoshi's change detector that identifies the "significant" changes in a scene to derive an object movement. If the segmentation mechanism fails to produce the action sequence intended, Teaching by Showing cannot correctly derive the intended code. Unlike this, our RPD system enables the demonstrator to correct the outcome of the segmentation mechanism using the RPD analyzing tool (see fig. 9). The strength of the RPD approach, however, is the induction component that allows the user to apply the generated program to many different problems of a certain class (see section 5.2). Another advantage is the capability to recognize control structures like loops (under development) and branches which is useful for real industrial tasks like loading pallets.

Research similar to that of Kuniyoshi et al. is performed by Ikeuchi and Suehiro (Ikeuchi et al., 1993). They call their approach *Assembly Plan from Observation* or *APO*. In contrast to Kuniyoshi, they use a video camera and a laser range finder. Furthermore, they have defined a more elaborated and *complete* model. The basic idea in APO is the detection of the so-called *face contact relations* between all objects and use them to derive unambiguous interpretations and generalizations of EOs. Since the model has to be able to completely explain every possible action only small problems can be solved with the APO approach.

5.2. RPD with generalized repetition

RPD with the possibility of generalizing the demonstrated programs has two advantages over RPD with direct repetition. First, RPD with generalized repetition could shorten programming times if large repetitive robot tasks like loading a pallet can be programmed by detecting loops from few example demonstrations of the repetitive steps. Second, programs with generalized repetitions can easily be adapted to new situations and, therefore, are interesting for the aspect of programming reusable code.

An application of RPD with generalized repetition was reported by R. Heise (1989) from the University of Calgary. Her system ETAR (Example-Based Task Acquisition in Robots) is able to generate programs for blocksworld constructions and, more recently, also programs

for placing objects on pallets or a conveyor belt (Heise, 1992). The programs contain variables, loops, and branches. Similar to our demonstration interface, Heise also does not operate with complex vision sensors but uses the master-slave robot setup EXCALIBUR that records 10 robot locations per second.

In order to extract the important data from the stream of robot locations, Heise defined a heuristic called “focus of attention” (FOA). This focus contains all location data in the vicinity of the objects to be manipulated (vicinity is the distance of the robot gripper to an object, a configurable value, typically three inch). All location data on a path between two vicinities are thrown away and replaced by one transport-motion command. Although the FOA criterion considers the context of actions to segment traces, it does not guarantee a user-intended segmentation. Wrong segmentations lead to problems during the derivation of a generalized program structure because the slightly different action sequences might not match. The RPD system presented here addresses this problem by enabling additional interaction with the RPD analyzing tool (see fig. 9) to correct the segmentation if necessary.

Generalization in ETAR is realized in two steps. In the first step, the “outer generalization” derives the program structure by grouping actions and searching for loops. Like in fig. 4, subsequences enclosed by gripper operations are identified in the action sequence. In each case, they define one semantical unit that describe the transfer of objects within the scene. After the subsequences have been identified, similar subsequences will be used to induce loops. Each loop generalizes a set of similar subsequences. The method used is similar to those applied in (Dufay & Latombe, 1984) or (Andrea, 1985). Structures within the subsequences are determined using the same method and induce further inner loops or branches. Unlike the RPD system, ETAR provides no mechanism to deal with ambiguities while clustering similar subsequences. Therefore, the derived program structure might not meet the user’s intention. The RPD system is using interaction (see section 4) with the user to ensure that the user-intended loops and branches are induced. The only other approaches to use additional interactions to ensure a program structure that meets the user’s intention are implemented as part of the PRODEGE+ graphics editor (Sassin, 1994) or METAMOUSE (Maulsby & Witten, 1993).

In the second step, ETAR performs the “inner induction” that generalizes the parameters of the EOs as well as the determination of the loop and branch conditions. This is done using an expensive generate&test algorithm which determines exactly one solution. Nevertheless, this method does not ensure that the solution meets the user’s intention. The RPD system presented does not only provide automated inner induction, but supports these steps by user interaction. This is similar to approaches in the graphics editor domain (Maulsby & Witten, 1993; Cypher, 1991; Sassin, 1994) or learning agents (Bocionek & Sassin, 1993; Bocionek, 1995). It results in safer robot programs that more probably meet the user’s intention. Furthermore, the RPD approach can be applied to a larger variety of problems because it is not restricted to a fixed set of basic functions as this is necessary for ETAR’s inner induction.

Another possible approach to cope with very few examples is explanation-based learning (EBL). It was explored by Segre in his ARMS system (Acquiring Robotic Manufacturing Schemata), a graphical simulation testbed for blocksworld and some peg-in-hole problems (Segre, 1988). ARMS uses EBL to derive a function schema from one example that

allows him the application to situations similar with respect to the object types and spatial relations, but different with respect to geometric locations and distances. ARMS is based on a sophisticated geometric reasoner that analyzes the start and goal states of a certain assembly.

6. Conclusions and future work

In (Milne, 1994), Milne uses the metaphor of two mountaineering pioneers: Both tried to climb a mountain. One uses a horizontal and the other one a vertical approach. The first one made more progress initially, but never reached the top because there was no easy way up. The second one had much more problems in the beginning, decided to use better tools and to improve his skills, and eventually made it to the top.

In contrast to many approaches in artificial intelligence, machine learning, and robot programming by demonstration, the system presented in this paper follows the “vertical” approach. A real robot with real sensory equipment is used for the demonstration of new examples as well as for the execution of the generated programs. The system architecture (as presented in fig. 2) and the knowledge representation in the presented planning language are both very modular and can easily be extended to enhance the system’s functionalities. Extensions that are currently under development and possible directions of future research will be outlined below.

6.1. Summary

In this article, an application of *Programming by Demonstration (PbD)* in the robotics domain was presented. The objective of this research is the exploration of possibilities to use machine learning for automated or semi-automated robot programming. The need for such programming environments actually comes from the industrial demand to lower the costs of production processes. With respect to the robot programming task within such processes the costs can be reduced significantly by incorporating PbD through:

- shortening programming and test cycles,
- enabling workers without or with little programming skills to perform the programming, and
- making robot programs (or program fragments) reusable for various applications.

In order to support this effort we equipped an industrial PUMA 260b manipulator with some sensors and a 6D bowl input device. Using this demonstrational interface all robot locations as well as some sensor data are recorded. The trace is first filtered to get rid of unnecessary intermediate locations. Then a segmentation takes place to find out meaningful subsequences which are classified into five classes of symbolic *Basic Operations (BOs)*. For solving this segmentation and classification task on the continuous demonstration data, subsymbolic machine learning techniques (*Time Delay Neural Networks*) are used.

After the transformation of the demonstration representation on the symbolic level the induction component is started to generalize and optimize the determined sequence of Basic Operations. Therefore, the planning language contains operations for branches and loops besides the BOs, all of those modeled as STRIPS-like operators. Moreover a variable concept is integrated. The postcondition for each operator and the whole demonstration is determined. By incorporating a man-machine interface, the user is asked questions about the intentions related to the demonstration. By analyzing the demonstration with respect to the user's intentions superfluous and conditioned subparts are detected. Superfluous manipulations are eliminated, whereas branches are generated whenever conditioned manipulations are encountered. With this method, supported by introducing variables by the user in a dialogue, optimization and generalization takes place. The resulting plan is stored for later usage in a knowledge base.

The execution module selects an appropriate and applicable plan to a given task and environment from the available ones. The selected plan is prepared for execution by instantiating its variables with respect to the given environmental configuration and is transformed into a sequence of executable operations. Finally, the prepared plan is translated to the specific robot control language of the desired robot system and executed.

6.2. Discussion

Although the automatic analysis of the recorded demonstration traces and the induction component's optimization and generalization modules perform very good, some potential weaknesses of the RPD system have to be discussed within this section.

First of all, the lack of external sensors limits the overall system performance. It must fully rely on the closed-world assumption introduced by STRIPS, is not able to autonomously build up and maintain a world model, and cannot execute a plan reactively. As outlined in the following section, external sensor data will be integrated in the PbD process to overcome this weakness.

Another potential drawback of the system is the use of STRIPS-like operators in the planning language. One might ask: "What is the benefit of RPD when the STRIPS operators, a world model, and a goal specification are already available? Simply apply a standard planning algorithm, and sooner or later it will produce a plan that solves the task!" Although this is true in principle, planning is an NP-hard problem which means that—given a sufficiently complex world model and goal specification—the off line planning will take *very* long. Therefore, in many cases it is much more comfortable and faster for the user to give a demonstration. Moreover, it is also reasonable to integrate a standard planner into the RPD system, i. e. to combine PbD and off line planning in order to make the best use of both methods.

Finally, the fact that the system's performance relies fundamentally on the successful user interaction might be seen as a drawback. Undoubtedly, a system which is able to learn and generalize autonomously by always guessing the user's intention right would be the ultimate PbD system. Unfortunately, a system with such capabilities will not be implemented in the near future. Today's ML algorithms often learn plausible concepts with respect to the input data that nevertheless do not reflect the user's intention. Obviously, this behavior cannot be

tolerated for robot programming systems that shall be applied in manufacturing or service domains. Therefore, the tight but comfortable integration of the user in the demonstration process supports safe control and is rather a necessity and advantage than a weakness of the presented system.

6.3. Future Research

To extend the presented approach to more complex applications including interactions of grasped parts with other objects is the challenge for future work. Particular extensions are:

- *Reactive behavior:* Until now, the robot can operate in a static, well-structured and well-known environment only where nothing unforeseen happens, because no external sensors are used. In the future, an on line planner and an exception handler which provide reactive behavior to the robot will be added to the system. Therefore, external sensors that permanently observe the robot's world, e. g. vision sensors, are going to be used.
- *New applications:* If the robot is able to react to specific actions in the world by using external sensors, new applications become possible, e. g. manufacturing and assembling operations. Moreover, new operators representing pre-learned low-level skills (see Baroglio et al., 1996; Kaiser et al., 1995) will be integrated in the planning language. This will enlarge the basic manipulation capabilities of the system allowing compliant motions, e. g. insertion operations.
- *Induction of loops:* Some of the systems presented in section 5 do support the detection of repetitively occurring demonstration parts and to induce loop structures from them. An extension that will enable the system presented here to induce loop structures, too, is currently under development. Again, the focus has been set on the generation of user intended loops, i. e. loops that do not only reflect a repetitively occurring structure, but also the user's intention.

Adding these extensions to the RPD system will improve its applicability and planning performance significantly. Finally, the result will be a system which might serve as a programming system in real-world domains like manufacturing or service environments (e. g. households etc.), supporting the user in programming even complex tasks easy and comfortable by demonstration.

Acknowledgments

The RPD prototype system has been developed and realized at the Institute for Real-Time Computer Systems and Robotics, Prof. Dr.-Ing. U. Rembold and Prof. Dr.-Ing. R. Dillmann, Department of Computer Science, University of Karlsruhe. We gratefully acknowledge the support of Jürgen Kreuziger and Michael Kaiser, who helped to develop the RPD system with many ideas and fruitful discussions. A special thanks to Michael Kaiser

for implementing and maintaining the robot's interface. We would also like to thank the unknown reviewers of this article who made invaluable suggestions to help us revise the first submission.

Notes

1. Despite these problems, *teaching by showing* is still the most popular method in industrial applications today, especially in mass production.
2. Noisy compared to how the motion vector sequence of the robot would look like if the motions would be executed automatically.
3. Object Groups are closely related to the Key Groups introduced in (Heise, 1989).

References

- Andreae, P. M. (1985). Justified Generalization: Acquiring Procedures from Examples. *Technical Report AI-TR-834*. Artificial Intelligence Laboratory, MIT.
- Atkeson, C. G., Aboaf, E. W., McIntyre, J., & Reinkensmeyer, D. J. (1987). Model-based robot learning. In *Proceedings of the 4th International Symposium on Robotics Research*.
- Baroglio, C., Giordana, A., Kaiser, M., Nuttin, M., & Piola, R. (1996). Learning Controllers for Industrial Robots. *Machine Learning*, same issue.
- Bocionek, S. (1995). Agent systems that negotiate and learn. *International Journal Human-Computer Studies*, 42, 265–288.
- Bocionek, S. & Sassin, M. (1993). Dialog-Based Learning (DBL) for Adaptive Interface Agents and Programming-by-Demonstration Systems. *Technical Report CMU-CS-93-175*. Carnegie Mellon University, Pittsburgh.
- Cypher, A. (1991). EAGER: Programming Repetitive Tasks by Example. In *CHI '91 Conference Proceedings* (pp. 33–39). New Orleans, Louisiana: ACM Press.
- Dufay, B. & Latombe, J.-C. (1984). An approach to automatic robot programming based on inductive learning. *International Journal of Robotics Research*, 3, 97–115.
- Fikes, R. E. & Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2, 189–208.
- Flaig, T., Neugebauer, J.-G., & Wapler, M. (1994). VR4RobotS: a New Off-line Programming System Based on Virtual Reality Techniques. In *Proceedings of the 25th International Symposium on Industrial Robots* (pp. 671–678). Hannover, Germany.
- Friedrich, H. & Kaiser, M. (1995). What can Robots learn from Humans?, In *IFAC Workshop on Human-Oriented Design of Advanced Robotic Systems* (pp. 1–6). Vienna, Austria.
- Heise, R. (1989). Demonstration Instead of Programming: Focussing Attention in Robot Task Acquisition. *Research Report 89/360/22*. University of Calgary.
- Heise, R. (1992). Programming Robots by Example. *Research Report 92/476/14*. University of Calgary.
- Ikeuchi, K., Kawade, M., & Suehiro, T. (1993). Towards Assembly Plan from Observation: Task Recognition with Planar, Curved and Mechanical Contacts. In *Proceedings of the IEEE/RJS International Conference on Intelligent Robots and Systems* (pp. 2294–2301). Yokohama, Japan.
- Kaiser, M., Giordana, A., & Nuttin, M. (1994). Integrated Acquisition, Execution, Evaluation and Tuning of Elementary Skills for Intelligent Robots. In *Proceedings of the IFAC Symposium on Artificial Intelligence in Real Time Control* (pp. 145–150). Valencia, Spain.
- Kaiser, M., Retey, A., & Dillmann, R. (1995). Robot skill acquisition via human demonstration. In *Proceedings of the International Conference on Advanced Robotics* (pp. 763–768), Barcelona, Spain.
- Kreuziger, J. & Hauser, M. (1993). A New System Architecture for Applying Symbolic Learning Techniques to Robot Manipulation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. Yokohama, Japan.
- Kuniyoshi, Y., Masayuki, I., & Inoue, H. (1994). Learning by watching: Reusable task knowledge from visual observation of human performance. *IEEE Transactions on Robotics and Automation*, 10, 799–822.

- Liu, S. & Asada, H. (1993). Teaching and training of deburring robots using neural networks. In *IEEE International Conference on Robotics and Automation* (pp. 339–345).
- Lozano-Pérez, T. (1983). Robot Programming. In *Proceedings of the IEEE*, 71, (pp. 821–841).
- Maulsby, D. L. & Witten, I. H. (1993). Metamouse: An Instructable Agent for Programming by Demonstration. In A. Cypher (Ed.), *Watch What I Do: Programming by Demonstration*. MIT Press.
- McKerrow, P. J. (1991). Introduction to Robotics. In *Electronic Systems Engineering*. Addison-Wesley.
- Milne, R. (1991). Building Successful Applications: The Wrong Way and the Right Way. In G. Barth et al. (Eds.) *KI-94 – Anwendungen der Künstlichen Intelligenz*. Springer.
- Münch, S., Kreuziger, J., Kaiser, M. & Dillmann, R. (1994). Robot Programming by Demonstration (RPD) – Using Machine Learning and User Interaction Methods for the Development of Easy and Comfortable Robot Programming Systems. In *Proceedings of the 25th International Symposium on Industrial Robots* (pp. 685–693). Hannover, Germany.
- Münch, S., Sassin, M. & Bocionek, S. (1994b) The Application of PbD Methods to Real-World Domains: Two Case Studies. In *Proceedings of the 7th Australian Joint Conference on Artificial Intelligence* (pp. 92–99). Armidale, Australia.
- Neubauer, W., Bocionek, S., Möller, M. & Rencken, W. (1992). Learning Systems Behavior for the Automatic Correction and Optimization of Off-line Robot Programs. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, Raleigh.
- Sassin, M. (1994). Creating user-intended programs with programming by demonstration. In *1994 Proceedings of the IEEE/CS Symposium on Visual Languages* (pp. 153–160). St. Louis, Missouri.
- Segre, A. M. (1988). *Machine Learning of Robot Assembly Plans*. Kluwer Academic Publishers.
- Thrun, S. B. & Mitchell, T. M. (1993). Integrating Inductive Neural Network Learning and Explanation-based Learning. In *Proceedings of the 13th International Joint Conference on AI* (pp. 930–936). Chambery, France.
- Ude, A. (1993). Trajectory Generation from Noisy Positions of Object Features for Teaching Robot Paths. *Robotics and Autonomous Systems*, 11, 113–127.
- Ude, A., Bröde, H. & Dillmann, R. (1994). Object Localization Using Perceptual Organization and Structural Stereopsis. In *Proceedings of the 3rd International Conference on Automation, Robotics and Computer Vision*, Singapore.
- Waibel, A., Hanazawa, T., Hinton, G., Shikano, K. & Lang, K. (1989). Phoneme recognition using time-delay neural networks. *IEEE Transactions on acoustics, speech and signal processing*, 328–339.