Towards Cognitive Robots: Building Hierarchical Task Representations of Manipulations from Human Demonstration

R. Zöllner, M. Pardowitz, S. Knoop and R. Dillmann
Institute for Computer Design and Fault Tolerance
Universität Karlsruhe (TH)
Karlsruhe, D-76128, Germany, P.O. Box 6980
{zoellner, pardow, knoop, dillmann}@ira.uka.de

Abstract—This paper deals with building up a knowledge base of manipulation tasks by extracting relevant knowledge from demonstrations of manipulation problems. Hereby the focus of the paper is on modeling and representing manipulation tasks enabling the system to reason and reorganize the gathered knowledge in terms of reusability, scalability and explainability of learned skills and tasks. The goal is to compare the newly acquired skill or task with already existing task knowledge and decide whether to add a new task representation or to expand the existing representation with an alternative. Furthermore, a constraint for the representation is that at execution time the built knowledge base can be integrated and used in a symbolic planner.

Index Terms—Modeling of Manipulation Tasks, Programming by Demonstration, Reasoning about Tasks and Skills

I. INTRODUCTION

Building intelligent robots has been a main topic for many years in the robotic research. Many approaches have been developed concerning different levels of intelligent control, learning strategies, planning and decision making embedded in various system architectures. In terms of building cognitive robots learning and reasoning are main abilities which have to be integrated on all levels of such systems.

Looking only on the manipulation facilities of cognitive robots, the acquisition of new skills and tasks is one major topic and gathering this knowledge from users through interaction or demonstration is the most intuitive way. But cognitive robots must also be able to reason about acquired knowledge and integrate it in the overall knowledge base of the system in order to fulfill requirements like reusability, scalability and explainability. Especially the last point enables robots to communicate with humans giving them on one hand informations about the task execution, missing information etc. and on the other hand adapting their internal representation through interaction. The internal representation of the tasks therefore is crucial, but the representation depends on all other components of the system and can’t be separated from them.

The remainder of this paper is organized as follows: In section II an overview of related works concerning programming by demonstration (PbD), task descriptions and symbolic planning is given. Section III describes briefly the knowledge acquisition of manipulation tasks from an user demonstration. Section IV addresses the hierarchical representation of manipulation tasks. In section V the integration of new task knowledge into the knowledge base using deductive reasoning is presented. Some drawbacks and so far not addressed problems are discussed in section VI.

II. RELATED WORK

Several programming systems and approaches based on human demonstrations have been proposed during the past years. Many of them address special problems or a special subset of objects only. An overview and classification of the approaches can be found in [1], [2]. Basis for the mapping of a demonstration to a robot system are the task representation and task analysis. Often, the analysis of a demonstration takes place observing the changes in the scene, described by using relational expressions or contact relations [3], [4].

Issues for learning to map action sequences to dissimilar agents have been investigated by [5]. Here, the agent learns an explicit correspondence between his own possible actions and the actions performed by a demonstrator agent by imitation. In [6] the authors concentrate on the role of interaction during task learning. They use multiple demonstrations to teach a single task. After generalization they use the teacher’s feedback to refine the task knowledge.

To generalize a single demonstration mainly explanation based methods are used [7], [8]. They allow for an adequate generalization taken from only one example (One-Shot-Learning). A generalized task description that is to be executed must be mapped onto an executable robot program. Several robot task representations have been proposed in the past. They can be roughly classified into programming languages including control structures and
declarative sequence- or tree-like descriptions of the task. An overview can be found in [9].

True robot programming languages are mostly built as extensions to standard programming languages like C (see Colbert[10]) or LISP (see RAPs [11][12], Frob [13]). Thus, runtime extension of tasks is not possible or only according to predefined rules. As it is not possible to extract control structures out of a single user demonstration, this approach is not feasible for a PbD scenario. Declarative task descriptions often rely on static task descriptions. These tree-like (e.g. TDL [14]) or procedural (see [15], [16]) descriptions can be extracted from user demonstrations and are closely related to PbD output action sequences. However, most existing declarative task descriptions have been designed for manual task compilation and do not provide extensive support for condition management, which is a prominent requirement for the execution of learned action sequences.

Given the whole set of basic operations including the relevant pre- and post-conditions, planning algorithms can be used to generate a sequential task description to reach a defined goal (e.g FF-planner [17]). As most goals allow for different paths, optimization criteria have to be applied to the planning algorithm. These criteria are not necessarily transparent to a human, which can result in the robot performing tasks in an unpredictable or even strange and uncanny way. Integration of task learning methods within a planning system can on the one hand help to solve these problems, and on the other hand enhance system performance by including learned task descriptions as atomic actions into the planning algorithm.

III. GENERATING ACTION SEQUENCES FROM USER DEMONSTRATIONS

The starting point for acquiring new knowledge is a Programming by Demonstration (PbD) system developed for many years at our institute. This section is giving a brief overview of the segmentation process and the manipulation task classes which can be learned by the system in order to describe the gathered information and the assumptions made during this process.

The main idea is to separate the acquisition of task knowledge from the execution of the task on a specific robot in order to achieve a robust task description from a vast external sensor system installed in a so called training center. The sensor environment uses two data gloves with integrated tactile sensors for gathering the finger joint angles and the forces applied on the fingers and palm, magnetic trackers for tracking the hand positions and two active stereo cameras for tracking the objects. For a detailed description of the demonstration environment refer to [18].

For ensuring a representation of robot invariant task knowledge the observed demonstration will be transformed to an abstract strips-like tree description called macro operator (MO) relaying on basic skills (primitives) called elementary operators (EO). An EO denotes an action that abstracts a sensory motor coupling like a grasp or a linear move etc. The EO’s build the hardware abstraction layer and have to be implemented on the robot system depending on the available sensors and actors.

The developed PbD system copes with intuitive demonstrations of manipulation tasks executed with both hands under the hypothesis that the user is doing all the changes in the environment (Closed world assumption). Furthermore it can process the following classes of manipulation tasks: transport operations (Pick&Place), device handling (like opening a drawer, pushing a button etc.) and tool handling (puring a glass of water, screwing etc.) (Figure 1).

![Fig. 1. Manipulation classes distinguished by the PbD system](image)

The phases of the PbD process, described in [19], are a segmentation of the sensor data followed by an analysis and interpretation step for identifying a sequence of EO’s which is abstracted and generalized to a macro operator (MO) in a further step.

In order to process the above manipulation classes the segmentation of the sensor data is done in three Phases:

- **Grasp segmentation:**
  Here a stable segmentation of grasp and release actions is performed.

- **Trajectory segmentation:**
  This phase extracts for each grasp and release segment an approach and dis-approach trajectory and fragments the whole hand trajectories in elementary move operations like linear, or circular segments.

- **Statistic segmentation:**
  For detecting object relations during grasp phases, statistic parameters are used in order to generate hypothesis on possible interactions using object-role-dependent probabilities ([20]).

The segmentation of the user demonstration uses a lot of background knowledge about the manipulation process and the environment, respectively the objects used in the demonstrated manipulation. In table I the parameters used for the segmentation are listed. The table shows the dependence of the parameters from the manipulation class. For
simple transport actions, in addition to the grasp type, only the trajectory of the objects is needed. Looking at device handling tasks, where the manipulation changes the internal state of the manipulated object (i.e., for a "open a door" action the state "door closed" changes to "door open") more information about the object has to be included in the process. The most complex manipulation in terms of detection is a tool handling since in this case the interaction between the manipulated objects determines the goal of the action. In this case more information about the object types and their functional roles is needed in order to detect and describe this kind of manipulations.

**TABLE I**

<table>
<thead>
<tr>
<th>Basic Skill</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport:</td>
<td></td>
</tr>
<tr>
<td>Grasp:</td>
<td>+ Object model (Type)</td>
</tr>
<tr>
<td>static</td>
<td>+ TCP velocity, joint angles</td>
</tr>
<tr>
<td>dynamic</td>
<td>+ forces</td>
</tr>
<tr>
<td>Move types</td>
<td>+ TCP trajectory analysis</td>
</tr>
<tr>
<td>Device handling:</td>
<td>+ +</td>
</tr>
<tr>
<td>Open doors, drawers</td>
<td>Move-axis, handle, state</td>
</tr>
<tr>
<td>Push/rotate buttons</td>
<td>Move-axis, handle, state</td>
</tr>
<tr>
<td>Tool handling:</td>
<td></td>
</tr>
<tr>
<td>Screwing</td>
<td>+ Functional Role</td>
</tr>
<tr>
<td>Pouring</td>
<td>&quot;Screw-able&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;pour in /out&quot;</td>
</tr>
</tbody>
</table>

The result of the segmentation step is a list of key points which are indicating possible starting points of EO’s. For generating an action (EO) sequence for each fragment a search for an instantiation of EO’s is made. The search is triggered by the information about the type of the key points and the probabilities of the key points. The proof for the instantiation of a EO requires an evaluation of several EO conditions stored in the EO’s. For example for instantiating a static grasp a neuronal net is used in order to classify the grasp and a positive classification will result in an instantiation of a certain static grasp (i.e., circular grasp).

**IV. BUILDING HIERARCHICAL TASK REPRESENTATIONS FROM ACTION SEQUENCES**

Representing manipulation tasks as pure action sequences is not flexible and also not scalable. Therefore a hierarchical representation is introduced in order to generalize an action sequence and to prune EO’s to more complex subtasks. Looking only on manipulation tasks the assumption is made that each manipulation consists of a grasp and a release action. To cope with the above specified manipulation classes, pushing or touching an object is interpreted as a grasp. The representation of grasping an object constitutes a "pick" action and consist of three sub-actions: an approach, a grasp type and a dis-approach (Fig. 2). Each of these sub-actions consists of a variable sequence of EO’s of certain types (i.e., for the approach and dis-approach the EO’s are move types and the grasp will be of type grasp). A "place" action is treated analogously.

Between a pick and a place operation, depending on the manipulation class, several basic manipulation operations can be placed (Fig. 3). E.g., a demonstration of the task "pouring a glass of water" consists of the basic operations: "pick a bottle", "transport the bottle", "pure in", "transport the bottle" and "place the bottle". A sequence of basic manipulation operations starting with a pick and ending with a place is abstracted to a manipulation segment.

The level of manipulation segments denotes a new abstraction level on closed subtasks of manipulation. In this context closed means that a manipulation segment ensures that both hands are free and that the environmental state is stable. Furthermore the synchronization of EO’s for left and right hands are included in the manipulation segments. Pre and post conditions describing the state of the environment at the beginning and at the end of a manipulation segment are sufficient for their instantiation. The conditions are propagated from the EO level to the manipulation segmentation level and are computed from the environmental changes during the manipulation. In parallel to the propagation of the conditions a generalization in terms of positions and object types and features is done.

A complex demonstration of a manipulation task is represented by a macro operator which is a sequence of manipulation segments. The pre and post conditions of the manipulation segments are propagated to a context and an effect of the macro operator. At execution time a positive evaluation of the context of a macro enables its execution and an error free execution leads to the desired effect in the environment.
V. BUILDING TASK DESCRIPTIONS INCLUDING ALTERNATIVES FOR CLOSED ACTIONS

Once the hierarchical representation for the task is built, the next step is to exploit the acquired knowledge and reproduce the user demonstration using a robot. Given only one demonstration the system is able to reproduce the task even in different environments if the macro can be instantiated, i.e. the object variables stored in the MO can be mapped to the existing objects in the environment and the spatial conditions are fulfilled. The reproduced task will sequentially execute the manipulation segments of the MO. There is no possibility to change the order of the manipulation segments which for some reasons would be appropriate given a certain environment constellation. The idea now is to provide the system with action alternatives and let it choose the most appropriate one. The alternatives are derived from different MO’s that have to be recorded in additional demonstration sessions. To establish indices to equivalent MO’s that could be executed alternatively, there has to be a measure of equivalence between MO’s.

Each time a new MO is recorded, its pre- and postconditions are calculated that are necessary to ensure proper execution. The set of pre- and postconditions is compared with the according set for every MO in the knowledge base of learned manipulation sequences. When a different MO is found with the same pre- and postconditions the system has discovered an alternative execution possibility.

To utilize the extra-knowledge of multiple demonstrations a flexible representation is needed that allows the executing system to choose from different actions and different execution orders. This representation has to incorporate knowledge about the necessary order of actions and which parts of the demonstrations are independent of others. The information, which action has to be executed before another one can be performed (precedence) and which operation can be scheduled anywhere in the execution (because it does not depend on any other), can be deduced from the precedence graph. This graph has to be built from all the MO’s representing the same task.

A. From Macro Operators to a precedence graph

To discuss the generation of a precedence graph from multiple macro operators let’s introduce an example: The demonstration is the task of laying a table with a plate, a saucer and a cup being put on the saucer. The action of placing the cup is dependent of placing the saucer because the saucer has to be on place before the cup can be put on it. So there is a chronological dependency between the saucer and the cup, saying that the saucer has to be placed before the cup can be put down on it. On the other hand, the manipulation of the plate is completely independent of the other two actions. It can be performed before the the saucer is placed, between the manipulations of the saucer and the cup or after the cup is put on the saucer.

Suppose in the first demonstration the plate is placed first, followed by the saucer and the cup (see top left in figure 4). Since there is only one (sequential) demonstration, the system is unable to reason about the independence of the placing of the plate from the saucer and the cup. So the precedence graph is the sequential concatenation of the elementary actions (depicted on the top right in figure 4).

Generally speaking, a single demonstration $D_1$ consists
of a sequence of manipulation segments \( s_1 \ldots s_n \), which implies a set of precedence rules \( P_1 \):

\[
D_1 = s_1 \iff s_2 \iff s_3 \ldots s_n \quad (1)
\]

\[
P_1 = (s_2|s_1) \land (s_3|s_1) \land (s_4|s_2) \ldots (s_n|s_{n-1})
= \bigwedge_{k=1..n,l=1..(k-1)} (s_k|s_l) \quad (2)
\]

where the term \( x|y \) means doing manipulation \( x \) after manipulation \( y \).

If the user performs another demonstration where he first puts the saucer on the table, followed by the plate and the cup, we are able to deduce the fact that the placing of the saucer does not depend on the placing of the plate. This results from the observation that in the first demonstration the plate is manipulated before the saucer and that this ordering is reversed in the second sequence. Therefore the sequential dependence in the precedence graph is resolved and the plate and the saucer are located on parallel branches in the graph (see figure 4). This means that each action can be performed independent of the other. However, note that the placing of the cup still depends on the placing of the plate. The system has only seen two demonstrations and in both of them the plate was manipulated before the cup, so there is no reason to assume that these operations are independent of each other.

Generally, this corresponds to adding a second demonstration \( D_2 \) with the same pre- and post-conditions but a different order of manipulation segments \( t_1 \ldots t_n \)

\[
D_2 = t_1 \iff t_2 \iff t_3 \ldots t_n
\]

with

\[
\exists f : [1..n] \rightarrow [1..n], f bijective : t_i = s_{f(i)}
\]

\[
P_2 = (t_2|t_1) \land (t_3|t_1) \land (t_4|t_2) \ldots (t_n|t_{n-1})
= \bigwedge_{k=1..n,l=1..(k-1)} (t_k|t_l) \quad (4)
\]

where \( t_1 \ldots t_n \) resembles to \( s_1 \ldots s_n \) but in different order.

To get the resulting precedence rules, the equation

\[
P_{new} = P_1 \lor P_2
\]

has to be evaluated.

At last a third demonstration is added placing the saucer and the cup first and the plate afterwards. Now the system can reason, that the placing of the plate is also independent from the placing of the cup and the precedence graph (see figure 4) consists of two branches with the saucer and the cup on one and the plate on the other. So the system has learned the independence of the plate from the saucer and the cup.

Thus, for \( r \) MOs that represent the same task, the precedence rules have to be set up for each MO, and then the overall precedence rules are given by

\[
P = \bigvee_{q=1..r} P_q
\]

which defines the precedence graph.

### Table II

<table>
<thead>
<tr>
<th>Demo 1</th>
<th>Demo 2</th>
<th>Demo 3</th>
<th>Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>S</td>
<td>S</td>
<td>C</td>
</tr>
<tr>
<td>S</td>
<td>P</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>C</td>
<td>(P ∧ S)</td>
<td>C</td>
<td>(P ∧ S)</td>
</tr>
<tr>
<td>C</td>
<td>(P ∧ S)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### B. Execution of Macro Operators

On the execution stage the system is able to exploit the information present in the precedence graph. If the system is given the command to lay the table with a plate, a saucer and a cup, the next step is to generate a sequence of actions that is valid in terms of not violating the precedence rules from the precedence graph. The selection can be made to take into account measures of optimality like execution time or energy-efficiency. Another criterion could be the availability of alternatives - the system should choose the action sequence so that there are many demonstrations for this actions available.

Suppose for any reason the system has decided to manipulate the saucer, the plate and the cup in this order, it comes to choosing the elementary actions for each manipulation. We denote the action of placing the plate, saucer, cup with \( P, S, C \) respectively. \( P \mid \emptyset \) means that there has been no action performed before \( P \) and \( C \mid (P \land S) \) that the plate and the saucer are placed before the cup is being placed. The three demonstrated action sequences are split into the single actions according to the first three columns of table II. The last column represents the action sequence to be performed.

To obtain the possible alternatives to choose for an action a simple search in this table has to be performed. For the execution of the placing of the saucer we are able to find two alternatives, one from the second and one from the first demonstration (These are printed in bold letters). For the second action to be executed, there is only one demonstrated action that fits at this point, the placing of the plate in demo 2. For the last action, placing the cup on the saucer, there are again two appropriate possibilities from the first two demonstrations. Using any optimization criterion the system now is able to generate an action sequence incorporating actions that the user has demonstrated in all three demonstrations, e.g. placing the saucer as demonstrated in demonstration 3, then placing the plate with the same trajectory as demonstrated in demo 2 and last placing the cup as performed in demo 1.

#### VI. Discussion

With the methods described above the system is put in the position to re-schedule user demonstrations. Dependencies between them can be extracted and independencies can be used to add some degrees of freedom to the execution order exploited for optimization.

First, when a new MO has been analyzed, it is compared to all existing MOs in the knowledge base regarding its pre- and postconditions. This MO can only be added to the existing precedence graph, if it contains exactly the same
set of (possibly permuted) closed manipulation segments. If the MO contains a different actions set having the same effects (e.g. moving saucer and cup independently in contrary to moving the saucer with the cup on top of it), it can not be added to the existing precedence graph and has to be stored as a different precedence graph for the same task.

Second, two MOs that contain a common subset of actions but perform different tasks according to the pre- and post-conditions, are not detected as similar. Therefore, the contained actions are not considered as alternatives, although it is theoretically possible. For example, laying the table with a plate, a saucer, a cup and a coffee pot is not found partially similar to setting the table with plate, saucer and cup.

These two shortcomings can be solved by also comparing the pre- and post-conditions of the manipulation segments comprised by the MO. Thus, all possible alternatives for each step can be found. However, this is a computationally expensive operation and can not be performed online. This is an issue not yet addressed in our work.

Increasing the granularity of the actions considered as alternatives could allow for better optimization at execution time. The system so far only deals with closed manipulation segments. It would be useful to extend this to the individual pick and place operations or even to a lower level, the approach and dis-approach trajectories.

Considering single pick or place actions (which are no closed segments) as alternatives can also aid in building precedence graphs for serialization of two-hand manipulations. Performing a task like setting the table with two hands in parallel can result in an action sequence like pick saucer (left), pick cup (right), place saucer, place cup. This can only be serialized if pick and place operations are considered as separate actions in the precedence graph.

Optimization at execution time, i.e. selection of alternatives, has not been addressed yet. Possible criteria can be standard optimization criteria like time or energy. In addition, it can be extracted from the user demonstrations, which alternatives the user prefers, or even which alternatives are preferred under certain circumstances.

VII. CONCLUSION

This paper presents an approach for integration of learned manipulation tasks into a knowledge base as well as a suitable representation of the tasks. For the integration, a reasoning step based on precedence relations is applied. Thus, the gathered knowledge is reorganized and structured on the level of manipulation segments, enabling an execution system to select from multiple alternative operations. As discussed in section VI, the presented approach denotes only one step towards building an optimal knowledge base of manipulation tasks. Potential enhancements which have to be addressed in future work have been discussed as well.

REFERENCES


