Representing, Learning, and Executing Operational Concepts

Volker Klingspor* **and Stefan Sklorz**

*University of Dortmund, Computer Science Dept., LS VIII, D-44221 Dortmund, Germany
{volker,sklorz}@ls8.informatik.uni-dortmund.de*

Abstract. *On the one hand side operational concepts enable the user to get information about what a robot has been done and on the other hand side, they enable the user to control the robot. In this way, they function as elements of a high level command language, easy to understand by human users. Operational concepts should be general enough to be applied in different but similar environments like office rooms even if the environment is totally unknown. They combine sensing and action of a mobile robot situated in the real world without the need of additional environmental knowledge. In this paper, we complete the representation of operational concepts and show, how this work is combined with results presented previously. Furthermore, we sketch the learning of concept descriptions to reach a high adaptivity of the robot and we suggest the application of these concepts for planning and control.*

Key Words. Operational concepts, mobile robots, combining sensing and action, adaptivity, inductive logic programming

1 Introduction

During the last years, mobile systems have become more and more powerful and hence more complex. Nevertheless, the interface between the human user and the robot is still tailor-made for the robot. So, the robots are mainly programmed by giving the exact locations of the positions the robot has to move to. In the best case, the map the robot uses for navigating consists not only of free and occupied areas, but also of identifier for special positions, i.e., landmarks. These landmarks can then be used by the user to command the robot. However, this kind of information must be known a priori, goal driven navigation in unknown environments is impossible with this approach. Our intention of commanding a robot is to use high-level concepts like doorways, walls, desks, or, more precisely, *operational concepts* like moving through a doorway or moving along a wall. How to build up a representation for such concepts, how to acquire them, and how to apply them will be presented in this paper.

Many approaches to command and control mobile systems base on the early work of Fikes and Nilsson [3]. This way of action planning is located on a cognitive high level allowing a human user to command the robot easily and to understand the generated plans. To be able to plan and to execute the plans, the environment must be known. The execution of the commands and the detection of objects are represented independent of the operators, usually they are explicitly programmed. So, the systems are not much adaptive and they cannot react to unforeseen events. To reach a higher adaptivity, the plans used for navigation can be acquired automatically while moving in the environment [2, 20], e.g., by detecting free and occupied areas. To enable the use of cognitive elements in these maps, like, e.g., an occupied area representing a desk, the classification of the perceptions is necessary. In those approaches, this work is mostly done by the user.

The weaknesses of the approaches described previously is the necessity of reasoning exclusively on a high and very complex level, leading to slow and unflexible systems. Behavior based systems [11, 10] are the contrasting approaches. Their behavior is determined by a set of very simple stimuli-response-rules, i.e., rules that trigger a specific action depending on the sensed pattern. The intelligent behavior of these systems is reached by an effect called emergency, i.e., that the different simple rules create a complex behavior when they work together (see [19] for a deeper discussion about emergent and hierarchical systems). In these approaches, no global knowledge about the environment is represented, thus they are very well suited for unknown environments. Because of the simple reactive rules, these systems can react very fast to sudden changes of the environment. However, the first of these systems are fully autonomous, there is no possibility for a user to influence the behavior during runtime. Additionally, these system cannot communicate in a human adequate way with the user, because they lack a cognitive level. By Firby, an approach is presented controlling different behaviors by switching on or off, resp., some of the stimuli-response-rules, to perform an externally requested behavior [4]. Steels presented an approach to classify the perceptions of a robot using a frame system and to control the behavior dependent on different internal states, which can also be requested from outside [18].

In our approach, we want to go one step further. One of the main issues is, that the cognitive elements can immediately be used for communication with the user. For this task, it is necessary that the concepts, used as elements of speech, span the whole hierarchy from high-level cognitive concepts to raw data gathered by the robot. In this way, the concepts can be used to classify the perceptions and actions and thus enables the user to get information about what the robot has been done in terms of high-level concepts. Moreover, the concepts are executable, i.e., the robot must be able to perform the action intended by the concept. Because of this capability we call them *operational concepts*, distinguishing them from the concepts usually occurring in machine learning literature.

In [8], two main aspects of operational concepts necessary to achieve these properties are presented in detail: First, the concepts must integrate action and perceptions, where actions are perception-oriented and perceptions are action-oriented. So, classification depends not only on perceptions, but also on the performed actions and thus, the success of the action. In this way, an object is only an instance of a concept, if the intended purpose of it is achieved.

Second, the features used for classifying the objects should depend directly on the real world data, they should by anchored in the environment. This problem encloses the hard problem of feature generation, because the quality of these basic features determines the powerfulness of the overall representation.

The concepts are represented on different levels of abstraction, the process of classification passes through multiple, differently detailed levels to eliminate faulty measurements and noise. The different elements of this representation hierarchy are described in Section 2, accompanied by some examples. To reduce the effort to create a new representation if the kind of environment, the kind of tasks, or the robot itself changes, we want to *learn* concept descriptions instead of modeling them. Learning will then lead to a high adaptivity of the system. Results of learning perceptual features are already presented [13], here we only describe the approach for learning action features and integrating concepts (Sect. 3). Since operational concepts will not only be used to classify perceived objects but also to control the robot, in Section 4, we present an approach to use these concepts for action planning as well as for execution and verification of plans. As usual, the paper ends with a short conclusion, summarizing the results and showing some weaknesses of our approach.

2 Representing Operational Concepts

Before we start to describe the representation of operational concepts, we sketch the framework in which this work is embedded.

A) The robot we use to apply the ideas about operational concepts is PRIAMOS, developed at the University of Karlsruhe [1]. PRIAMOS moves on four separately controlled Mecanum-wheels, allowing the robot to move in three degrees of freedom. 24 sonar sensors are mounted on the robot, three of them at each side and three at each corner, all in the same height. Besides other data, at each time of sensing, PRIAMOS delivers the sensed distances, the current position, and the current orientation relative to its coordinate system.

B) During a trace, the sensors of PRIAMOS get measurements characterizing the environment in their own way of sight. To conclude from these measurements the composition of the environment, it is necessary to find typical patterns, also called features. The sensing of these features during the same interval of time or with a short delay during an action of the robot allows to conclude objects. Morik, Klingspor, and Rieger describe the representation of these features on different levels of abstraction and how to learn descriptions for them [13, 9]. In particular, they distinguish *basic perceptual features*, *sensor and sensor group features*, and *action-oriented perceptual features*, each of them defined in terms of the next lower level. Action-oriented perceptual features represent objects of the environment, considering the path of the robot relative to this object. They embody the perceptual part of operational concepts in contrast to the action part and the integrating part presented in this work. Examples for perceptual features are: through_door/6, along_door/6, in_front_of_wall/4, and in_front_of_door/4.

The goal pursued while developing operational concepts is to combine the action of a robot with the perceptions got while executing these actions. This combination shall allow to identify the performed actions and the according objects as well as to execute and control actions. The following requirements have led our work:

- For each operational concept, the situation in which it is applicable (the precondition) and the situation in which its application usually ends (the postcondition) must be represented.
- To use concepts for planning, pre- and postconditions must be suitable to choose the next sensible concept dependent on the effect of the current one.
- The name of the concept must reflect the intended effect of its application to allow the usage of concepts as high-level elements of a command language.
- Descriptions for operational concepts and the features at all intermediate levels should be acquired by machine learning methods.

As far as the last item is concerned, note that we distinguish *learning phases* and *performance phases*. During the learning phase, we present PRIAMOS examples of the concepts in a known environment. From these examples, the system learns general descriptions of the concepts. In the performance phase, the learned concepts can be applied in different, unknown environments.

In this section we present the representation of *basic actions*, which are incrementally computed from the data delivered from the robot during its trip. We show how to combine basic actions and action-oriented perceptual features to integrated features, *perception integrating action features*. They represent actions of the robot in dependency on the perceived objects. In this way, they build the basic elements to represent the operational concepts themselves. Fig. 1 shows the overall hierarchy.

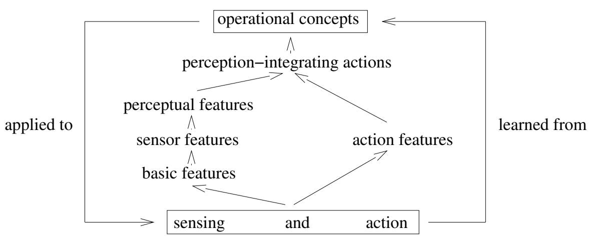

Figure 1: The representation hierarchy

Basic actions serve two tasks. During the learning phase, they are incrementally computed from the data delivered by the robot. During the performance phase, they trigger the elementary operations of PRIAMOS to control the robot. Basic actions are represented by the following predicates:

stand(Trc,T_1,T_2),
move(Trc,T_1,T_2,Speed,RDir),
a_move(Trc,T_1,T_2,Speed,ADir),
rotate(Trc,T_1,T_2,RSpeed,RODir).

The variable Trc represents the identifier of the considered trace, T_1 and T_2 describe the first and the last instant of time of the represented action. RDir and ADir describe the relative or absolute direction of the action, i.e., left, right, front, back. The variables Speed and RSpeed represent classes of speed for movements and rotations. RODir specifies the direction of a rotation.

Accordingly, for movements we designated two predicates. The first one, move/5, represents the direction of the the movement *intrinsically*, i.e., dependent on the current direction of motion and independent of a specific a priori given front of the robot. Starting with a movement, the direction the robot moves in becomes its front. If the robot rotates, the front of the robot does not change. At the level of operational concepts, this kind of representing directions yields the advantage that, e.g., the denoted direction *right* in the command *move through the doorway and then go to the right*, is independent of the particular orientation of the robot.

To transform the relative direction specified by a basic action move into elementary operations, a computation of an absolute direction according to the reference system of the robot is necessary. This absolute direction is represented by the a_move/5-predicate. Thus, we use the computation from a_move to move during the learning phase and vice versa during the performance phase. This transformation as well as the computation of a_move is easy and is therefore not described in this paper, details can be found in [16]. Fig. 2 illustrates the difference, the black spot at the robot marks its absolute front.

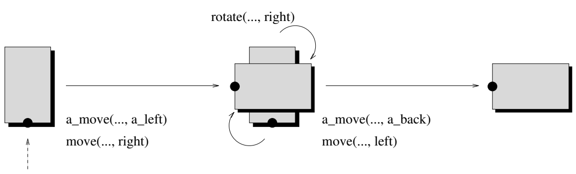

Figure 2: Directions of movements

At the next higher level of the hierarchy, perception integrating actions are represented. Before describing them in detail, we present two examples, illustrated in Fig. 3.

The feature p_moving/7 characterizes a parallel movement along the perceived object. The fact

p_moving(t3,13,25,slowly,front,along_door,right).

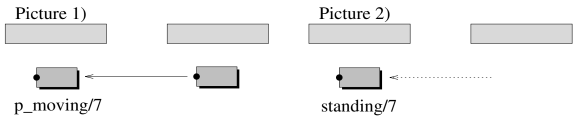

Figure 3: perception integrating action features

e.g., represents a slow movement along a doorway, perceived by the right side sensors during the time interval [13,25] (Fig. 3, Pict. 1). Picture 2) shows a situation, in which the robot is standing in front of a wall after having perceived a movement along a doorway. This situation is represented by the fact

standing(t25,25,29,in_front_of_wall,right,large_side,
along_door).

standing/7 can be used as a memory for the previous perception. This knowledge is necessary in some planning situations, e.g., if the robot shall move through the door, it must previously has been moved along a door. In particular, we distinguish between four predicates to represent action-integrating perceptual features:

standing(Trc,T_1,T_2,Perc,PDir,PSide,LPerc),
rotating(Trc,T_1,T_2,RSpeed,RODir,Perc,PDir),
moving(Trc,T_1,T_2,Speed,RDir,Perc,PDir),
p_moving(Trc,T_1,T_2,Speed,RDir,Perc,PDir).

In addition to the previously described variables, these predicates specify further arguments. Perc defines the perception the robot got during the action, i.e., the name of an action-oriented perceptual feature. LPerc states the last perception before the current action. The direction in which the object was perceived is defined by PDir, PSide additionally states whether this side was a large one or a small one of the robot. This information is necessary in situations, where the next action can only be performed with the small side oriented to an opening, e.g., if the robot should move through a doorway. If the wide side is oriented towards the object, the robot must perform a rotation first.

Perception-integrating action features are characterized by rules combining basic actions and action-oriented perceptual features. How these rules can be learned is described in Sect. 3, as an example, we present some intended rules here:[1]

per._perc(Trc,T_0,T_1,Perc,PDir,PSide,Orien) &
pt._perc(Trc,T_1,Perc1,PDir1,PSide1) &
stand(Trc,T_1,T_2)
→standing(Trc,T_1,T_2,Perc1,PDir1,PSide1,Perc).

move(Trc,T_1,T_2,Speed,RDir) &
pt._perc(Trc,T_2,Perc,PDir,PSide)
→moving(Trc,T_1,T_2,Speed,RDir,Perc,PDir).

move(Trc,T_1,T_2,Speed,RDir) &
per._perc(Trc,T_1,T_2,Perc,PDir,PSide,parallel)
→p_moving(Trc,T_1,T_2,Speed,RDir,Perc,PDir).

The predicates period_of_time_perception/7 and timepoint_perception/7 are used to distinguish features perceived during a time interval of arbitrary length, e.g., along_door, and features perceived at a single time, in_front_of_door. Both kinds of features are described by transformation rules, converting a predicate name into a constant argument of one of the two predicates:

in_front_of_door(Trc,T_1,PDir,PSide)
→pt._perc(Trc,T_1,in_front_of_door,PDir,PSide).

through_door(Trc,T_1,T_2,PDir,PSide,Orien)
→per._perc(Trc,T_1,T_2,through_door,PDir,PSide,Orien).

The highest level of the hierarchy represents operational concepts themselves. As described above, pre- and postcondition of the operator must be specified by their definition, and, additionally, pre- and postcondition should allow the linking of the different concepts to enable planning. These requirements yield in descriptions of the following structure:

<standing> &
<moving> | <p_moving> | <rotating> &
<standing>
→ <operational concept>

Some of the intended concepts need additional background knowledge to determine direction information. This background knowledge is represented by a set of facts characterizing different relations between directions. The predicate bg_opposite_direction(Dir,NDir), e.g., relates two opposite sides of the robot, e.g., left and right.

To extend the reader's comprehension, we explain a conceivable definition of the concept move_in_front_of_door in detail:

standing(Trc,T_1,T_2,in_front_of_wall,PDir,PSide,
along_door) &
bg_opposite_direction(PDir,PDir1) &
moving(Trc,T_2,T_3,Speed,back,in_front_of_door,PDir1)
standing(Trc,T_3,T_4,in_front_of_door,PDir1,PSide,
in_front_of_door)
→move_in_front_of_door(Trc,T_1,T_3,T_4).

The example is illustrated in Fig 4. The precondition of this concept is to stand in_front_of_a_wall and having perceived the feature along_wall during the last movement. This situation, shown in Pict. 1), is described by the first premise of the

[1] In the following, time-point perceptions are abbreviated by pt._perc., period-of-time perceptions by per._perc.

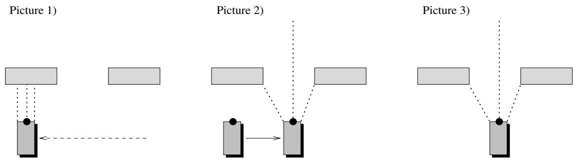

Figure 4: The concept move_in_front_of_door

rule. To move in front of the door, the robot must move backwards until it perceives that it stands centered in_front_of_the_door. This action, Pict. 2) in Fig. 4, is represented by the third premise. Because the direction of the movement has changed, the doorway is perceived in the direction opposite to the one in which the wall has been perceived previously. This transformation is made by the background fact in the second premise. After having perceived the doorway, the robot stops (Pict. 3), the fourth premise of the concept becomes true, and the concept is achieved. How to apply such a concept exactly, is described in Sect. 4.

In addition to the concept describe above, similar concepts are necessary for navigating in office environment. To solve tasks like moving through doors the following concepts are imaginable: move_closer_to_wall/4, rotate_in_front_of_wall/4, move_parallel_in_corner/4, rotate_in_corner_to_front_wall/4, move_along_door/4, move_through_door/4. The meaning of the concepts can easily be fixed by their names. In the following section, we describe how to learn descriptions at the different levels. In Sect. 4 we show our ideas on using concepts for planning, execution, and control.

3 Learning Operational Concepts

In the previous section, we described a representation of operational concepts, which is compatible with the representation of perceptual features described by [13]. The basic features of this representation can be delivered incrementally during the trip of the robot. In this section, we show how to learn descriptions for perception-integrating action features and operational concepts, using logic based inductive learning. Learning perceptual features is described in [13, 9].

The learning algorithm used is RDT [7], one of the algorithms of the area *inductive logic programming* (ILP) [14]. ILP-algorithms can be divided in two main groups, dependent on their objective. Algorithms of the first group try to induce a concept description from a set of positive and negative examples of the concept and additional background knowledge. The description must cover all positive examples and none of the negative ones [15]. Algorithms of the second group try to find regularities in a set of formulas. In contrast to the first group, these algorithms do not necessarily cover all examples of a concept [5]. These views have in common restricted first order logic as representation formalism. We apply RDT in the way described by Helft [5], and we only learn from positive examples[2].

The learner RDT is integrated in the knowledge acquisition tool MOBAL [12]. It uses a extended function free horn logic as representation formalism, also allowing negated literals. A special feature of RDT is the use of rule schemata, i.e., formulas with predicate variables instead of predicate symbols. They define sets of formulas of the same syntax. In this way, the rule schemata define the hypotheses space of RDT. The learning tasks can now be formulated:

LT 1: Learning Percept.-Integr. Action Features
Given:
- Examples for basic action features
- Examples for action-oriented perceptual features
- Rules about pt._perc. and per._perc.
- Examples for perception-integrating action features
- Rule schemata

Goal:
- Rules characterizing perception-integrating action features

LT 2: Learning Operational Concepts
Given:
- Examples for perception-integrating action features
- Background knowledge to transform given directions
- Examples for operational concepts
- Rule schemata

Goal:
- Rules characterizing operational concepts

For the experiments described here, five training traces of the robot are used. Fig. 5 displays one of them. This trace can be described by the operational concepts:

```
move_parallel_in_corner(t201,1,13,17).
rotate_in_corner_to_front_wall(t201,13,26,30).
move_along_door(t201,26,52,55).
move_in_front_of_door(t201,52,69,72).
rotate_in_front_of_door(t201,69,81,83).
move_through_door(t201,81,109,112).
```

We generated these examples manually using a graphical interface that visualizes the details of the trace and enables the user to edit the example set. The automatical generation of examples driven by the interface is a further issue of our work. Fig. 6 shows the visualization component of the interface.

[2]Under specific conditions, RDT can also be used as a learner in the sense of Plotkin [15].

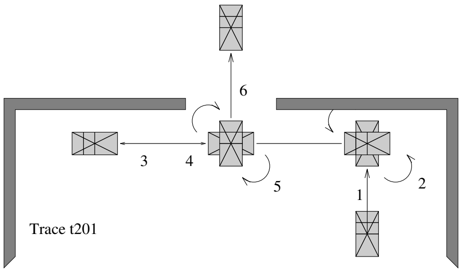

Figure 5: Example trace 201

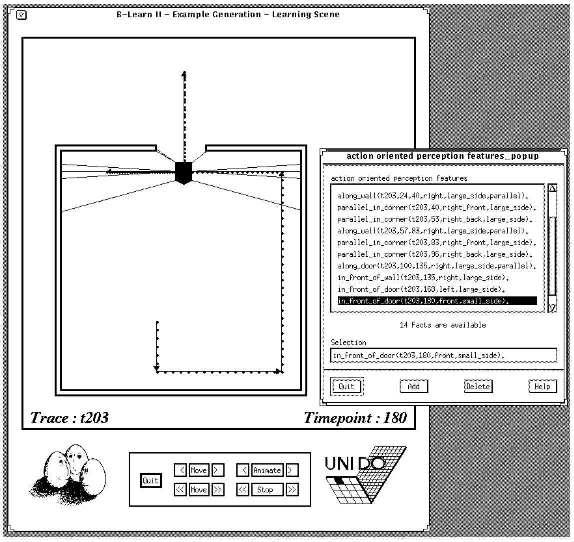

Figure 6: The human computer interface

Since we have given examples for each of the levels, independent of the rules to be learned, the two learning tasks can be solved in parallel. For each one, we performed three learning passes with different confirmation criterions. The confirmation criterions control the learning process by determining which hypotheses are acceptable. The different criterions we used are:

CC1: A hypothesis is acceptable, if it covers at least one positive example of the goal concept.

CC2: A hypothesis is acceptable, if it covers at least one positive example, but it does not derive any further, i.e. unknown, facts.

CC3: A hypothesis is acceptable, if it covers at least three positive examples, and at least 50 % of all facts derived by this rule are positive examples.

The tables 1 and 2 show the results of learning. The first column presents the number of learned rules, the second one the total number of examples, i.e., the number of input and derived facts. The third column shows the number of input examples, the fourth the number of facts derived by the learned rules. The percentage of covered input examples is presented in column five.

	#rules	#exampl.	#input	#deriv.	%cover.
perception integrating action features					
CC1	6	98	65	98	100%
CC2	4	65	65	43	66.15%
CC3	4	65	65	43	66.15%

Table 1: Quality of the rules learned for integrating action features

	#rules	#exampl.	#input	#deriv.	%cover.
operational concepts					
CC1	26	30	30	30	100%
CC2	19	30	30	30	100%
CC3	12	30	30	27	90%

Table 2: Quality of the rules learned for operational concepts

The results of learning perception-integrating action features show, that the coverage of the rule set learned with CC1 is much better than the coverage of the two other rule sets. With CC1, exactly two more rules are learned, one for standing and one for moving. Both rules are rejected by CC2 and CC3, because they derive too many additional facts. This problem can be solved by using more example traces.

The influence of our small learning set is even more relevant for learning operational concepts. For some of the concepts, we can present only a single example. Clearly, these concepts cannot be learned with the third confirmation criterion. We assume the coverage of the examples to be better when we use more examples.

Since the two learning tasks are performed independently, we have not yet given a statement about the quality of the whole rule set. Table 3 shows the results of applying *all* rules to the input data, i.e., the perceptual features and the basic actions. The poor coverage of the concepts for the rules set learned with CC2 and CC3 can be traced back to the missing rules for standing and moving. Especially standing is used very often in the descriptions of operational concepts, the missing facts thus result in few examples that can be covered. This insight reflects the results we also

	#rules	#exampl.	#input	#deriv.	%cover.
operational concepts					
CC1	26	30	30	30	100%
CC2	19	30	30	4	13.33%
CC3	12	30	30	4	13.33%

Table 3: Performance of all learned rules

got from earlier tests. Because of the multi-level representation hierarchy, it is in most cases better to learn more rules also covering faulty situations, than to learn few but correct rules. The faulty facts are mostly eliminated when climbing the hierarchy.

4 Planning and Control

To plan actions, an operator must not only consist of the situation in which it is applicable, but also of the situation, that arises from applying it, to be able to chain multiple operators. In the STRIPS-approach [3], operators are presented as structures with a list of preconditions, of facts to be deleted, and facts to be added; add- and delete-list define the new situation. In that context, action planning is defined to find a sequence of operators, the use of which satisfies the given goals, i.e., they occur in the add-list of an operator of the sequence without being deleted afterwards by another operator. Such a plan is usually found by goal-directed search, starting with the list of open goals and trying to solve each goal.

In our context, precondition and delete-list of a STRIPS-operator correspond to the first premise of the rule defining an operational concept and the add-list corresponds to the third premise. The action itself is not represented by STRIPS-operators. In contrast, operational concepts represent them by the second premise.

In principal it is easy to find a plan consisting of operational concepts, because the applicability of a concept can be tested by unifying its precondition with the postcondition of the previous operator. So, the algorithm starts with the goal concept thereby and gets the first precondition to be satisfied. By unifying this precondition with the postconditions of the existing concepts, we get all possible predecessors the execution of which will fulfill the precondition of the goal concept. By applying this step recursively for the predecessors until the precondition of a concept can be unified with the current state — which is in general a basic action and a corresponding perception — we get a sequence of operational concepts which is executable and reaches the goal. This very intuitive way of *concept based planning* is, unfortunately, too weak in the domain of navigating in unknown environments, because of the uncertainties that can arise. We will be back on this issue later in this section, but first, we suppose that a planner of the previously described kind has found a plan, represented as a sequence of operational concepts. This plan should now be executed, i.e., each of the concepts of the plan must be executed. We explain the execution with an example. Suppose, the robot is standing in front of a doorway and it shall move through it, i.e., it shall execute the concept move_through_door. This concept is defined by the rule:

standing(<u>Trc</u>, <u>T_1</u>, T_2, in_front_of_door, <u>PDir</u>, small_side, <u>LPerc</u>) &
p_moving(<u>Trc</u>, T_2, T_3, slowly, <u>PDir</u>, through_door, right_and_left) &
standing(<u>Trc</u>, T_3, T_4, in_front_of_door, back, small_side, through_door)
$\rightarrow$ move_through_door(<u>Trc</u>, <u>T_1</u>, T_3, T_4).

The first premise, standing(...), is already satisfied, its arguments PDir, Trc, T_1, and LPerc are instantiated according. To ease their understanding, instantiated arguments are underlined. A usual PROLOG-interpreter would now try to verify the second premise, p_moving(...), defined, e.g., by the following rule[3]:

move(<u>Trc</u>,T_2,T_3,<u>Speed</u>,<u>RDir</u>) &
per._perc.(<u>Trc</u>,T_2,T_3,<u>Perc</u>,<u>PDir</u>, PSide,parallel)
$\rightarrow$ p_moving(<u>Trc</u>,T_2,T_3,<u>Speed</u>,<u>RDir</u>,<u>Perc</u>,<u>PDir</u>).

So first, move(...) must be verified. move is a basic action, i.e., a predicate which could be proven by triggering an elementary operation of the used robot, here to move slowly in the specified direction. At the moment the robot starts moving, the variable T_2 gets the current time. Then, the interpreter waits for getting the perceptual feature per._perc.(...) which will be derived from the measurements as soon as possible[4]. As soon as this feature is delivered, the argument T_3 is instantiated. To stop the command move_through_door, its third premise, standing(...), must now be proven. standing is defined by the rule:

pt._perc.(<u>Trc</u>, <u>T_3</u>,<u>Perc</u>,<u>PDir</u>,<u>PSide</u>) &
stand(<u>Trc</u>,<u>T_3</u>,T_4)
$\rightarrow$ standing(<u>Trc</u>,<u>T_3</u>,T_4,<u>Perc</u>,<u>PDir</u>,<u>PSide</u>,<u>LPerc</u>).

At the moment the robot perceives that it is standing in front of the doorway, perceiving the opening at its rear, pt._perc.(...), the interpreter tries to satisfy stand(...), again a basic action and thus a predicate triggering an elementary operation, namely stopping. The concept move_through_door is now executed and at the same time its execution is verified, because the trace is classified as an instance of the concept.

To reduce the effort for proving the goal concept during runtime, a sequence of basic actions and corresponding perceptual features can be generated a priori. To do so, the list of concepts of the

[3] In the following, time-point perceptions are abbreviated by pt._perc., period-of-time perceptions by per._perc.

[4] Action-oriented perceptual features are always delivered by forward inferences, all other features can be derived forward (for classification) or backward (for triggering an action).

plan must be proven under the condition that all basic actions and action-oriented perceptual features are always true. During the prove, they are gathered including the variable bindings needed for the prove. In Fig. 7, a short sequence of operational concepts and the corresponding sequence of basic actions and perceptual features are shown. The features above each concept correspond to its precondition, the ones beside a concept to its action, and below to the postcondition. Note, that each two concepts share their pre- and postcondition.

world state	stand(Tr,T_1,T_2) parallel_in_corner(Tr,T_1,left,large_side)
move_along-door	move(Tr,T_2,T_3,med_speed,front) along_door(Tr,T_2,T_3,left,parallel) in_front_of_wall(Tr,T_3,left,large_side)
	stand(Tr,T_3,T_4) in_front_of_wall(Tr,T_4,left,large_side)
move_in_front-of_door	move(Tr,T_4,T_5,slow_speed,back) in_front_of_door(Tr,T_5,right,large_side)
	stand(Tr,T_5,T_6) in_front_of_door(Tr,T_5,right,large_side)
rotate_in_front-of_door	rotate(Tr,T_6,T_7,slow_speed,left) in_front_of_door(Tr,T_7,back,small_side)
	stand(Tr,T_7,T_8) in_front_of_door(Tr,T_8,back,small_side)
move_through-door	move(Tr,T_8,T_9,slow_speed,back) through_door(Tr,T_8,T_9,left,parallel) in_front_of_door(Tr,T_9,back,small_side)
	stand(Tr,T_9,T_e) in_front_of_door(Tr,T_9,back,small_side)

Figure 7: A plan and the corresponding features.

To execute this sequence, nothing unexpected must arise, because the system cannot react to those situations. Additionally, often it is not possible to specify the plan a priori, e.g., if it is not known how many walls the robot must follow before finding a door. How to solve this problem is not yet developed. However, we want to present our ideas and discuss them.

The first way is to search not only for one plan but for all plans up to an upper bound. These plans can be represented as a directed graph, the choice which successor should be executed in the specific situation is then determined by the current perception. Since some of the plans can be infinite or may contain loops — e.g., again and again moving along a wall and rotating in the corner — parts of the plans can be represented as cycles in the graph. The main problem for this approach is to find these cycles and to generate the cyclic, directed graph from a list of plans. Additionally, in this approach, the system cannot react to unforeseen events; to represent all exceptions that can be foreseen unnecessarily enlarges the graph.

The second way to handle uncertainties is to generate only the shortest plan and to execute this plan as long as possible, in the best case completely. At the moment the perception of the robot does not correspond to the perceptions expected by the plan, the system creates a new plan, starting with the current situation. In this way, the robot can easily react to unforeseen events and in totally unknown environments. We call this kind of planning *incremental planning*. It can be seen as an approach in between classical planning and behavior based systems.

Incremental planning, however, is expected to need much time during the execution phase. In our example, the system must search for a new plan whenever it moves into a corner without having perceived a door. Thus, a combination of both of the described approaches is in mind to be implemented, i.e., we intend to represent plans which are able to reach the goal if nothing unexpectable happens by a cyclic graph, and the system only re-plans in unforeseeable situations. The graphs found by the first approach can be stored and used as macro operators in further situations, corresponding to the work on learning macro-trajectories [17], but on a conceptual level.

5 Conclusion

In this paper, we presented an approach to represent actions and to integrate them with perceptions to define operational concepts in a first order representation formalism. The represented concepts can be used as elements of speech for communicating with a robot on a human-adequate level without loosing the contact to the environment in which the robot is embedded. We have shown how to learn concept descriptions to enlarge the adaptivity of robot applications. And we proposed an approach for action planning and to "compile" and execute the generated plans, principally only using a PROLOG-interpreter.

Clearly, the approach is far away from being fully developed. To show the effectiveness of learning, we have to perform further learning tests, using a higher number of more difficult example traces. Furthermore, we must develop the planning approach, e.g., solving the question what happens, if a concept is defined by multiple rules with the same or different premises.

One of the weaknesses of our approach is the fixed set of basic actions and the chosen set of basic perceptions, only applicable for a robot using only one degree of freedom at once. With this set, our system cannot reach the flexibility and the robustness of behavior based systems. However, coupling both approaches seems to make sense. In [6], a possible coupling of different learning approaches for different navigation tasks, inter alia the approach presented here, is described.

Acknowledgement

This work is partially funded by the European Community under the project B-Learn II (P7274) and the Ministry for Sciences and Research of the German federal state Nordrhein-Westfalen.

We would like to thank our advisor Katharina Morik for her ideas and suggestions. We thank Joachim Hertzberg and the members of our machine learning unit for the discussions and the many questions they asked. Additionally, thanks to Stephanie Wessel for reading drafts of this paper.

References

[1] Rüdiger Dillmann, Jürgen Kreuziger, and Frank Wallner. PRIAMOS – an experimental platform for reflexive navigation. In Groen, Hirose, and Thorpe, editors, *IAS-3: Intelligent Autonomous Systems*, chapter 18, pages 174–183. IOS Press, 1993.

[2] A. Elfes. A sensor-based mapping and navigation system. In *Proc. of the IEEE Intern. Conf. on Robotics and Automation*, 1986.

[3] Richard E. Fikes and Nils J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4):189–208, 1971.

[4] R. James Firby. Building symbolic primitives with continuous control routines. In James Hendler, editor, *Proc. of the first Intern. Conf. on AI Planning Systems (AIPS-92)*, pages 62–69, San Mateo, CA, 1992. Morgan Kaufmann.

[5] Nicolas Helft. Induction as nonmonotonic inference. In *Proceedings of the 1st International Conference on Knowledge Representation and Reasoning*, 1989.

[6] M. Kaiser, V. Klingspor, J. del R. Millán, M. Accame, F. Wallner, and R. Dillmann. Achieving intelligence in mobility – incorporating learning capabilities in real-world mobile robots. *IEEE-Expert (special track on Intelligent Robotic Systems)*, 1995. to appear.

[7] Jörg-Uwe Kietz and Stefan Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. In Muggleton [14], chapter 16, pages 335–360. Also available as Arbeitspapiere der GMD No. 503, 1991.

[8] Volker Klingspor and Katharina Morik. Towards concept formation grounded on perception and action of a mobile robot. In *Proc. of the 4th Intern. Conference on Intelligent Autonomous Systems*, 1995. In press.

[9] Volker Klingspor, Katharina Morik, and Anke Rieger. Learning operational concepts from sensor data of a mobile robot. (submitted to Machine Learning Journal), September 1994.

[10] Maja J. Mataric. *Interaction and Intelligent Behavior*. PhD thesis, Artificial Intelligence Laboratory, Massachusetts Institute of Technology (MIT), Cambridge, Massachusetts, 1994.

[11] Jean-Arcady Meyer and Stewart W. Wilson, editors. *From Animals to Animats – Proceeding of the First International Conference on Simulation on Adaptive Behavior*, MA, 1991. The MIT Press.

[12] K. Morik, S. Wrobel, J.-U. Kietz, and W. Emde. *Knowledge Acquisition and Machine Learning – Theory, Methods, and Applications.* Academic Press, London, 1993.

[13] Katharina Morik and Anke Rieger. Learning action-oriented perceptual features for robot navigation. In Attilio Giordana, editor, *Workshop notes: Learning Robots of the ECML-93*, 1993. Also available as Research Report 3, Univ. Dortmund, Informatik VIII, D-44221 Dortmund.

[14] Stephen Muggleton. *Inductive Logic Programming.* Number 38 in APIC series. Academic Press, London, 1992.

[15] Gordon D. Plotkin. A further note on inductive generalisation. *Machine Intelligence*, 6:101–124, 1971.

[16] Stefan Sklorz. Repräsentation operationaler Begriffe zum Lernen aus Roboter-Sensordaten. Master's thesis, Universität Dortmund, 1995. in German.

[17] Horst Spandl and Knut Pitschke. Lernen von Makro-Trajektoren für einen autonomen, mobilen Roboter. *Künstliche Intelligenz*, 1:12–16, 1991.

[18] L. Steels. Emergent frame recognition and its use in artificial creatures. In *Proc. of the 12 th IJCAI*, pages 1219–1224, Sidney, Australia, 1991.

[19] Luc Steels. Towards a theory of emergent functionality. In Meyer and Wilson [11], pages 451–461.

[20] Frank Wallner, Michael Kaiser, Holger Friedrich, and Rüdiger Dillmann. Integration of topological and geometrical planning in a learning mobile robot. In *Proc. of IROS-94*, 1994.