Learning Action-oriented Perceptual Features for Robot Navigation

LS–8 Report 3

Katharina Morik
Anke Rieger
Univ. Dortmund, LS VIII
D-44221 Dortmund [1]

Dortmund, April 26, 1993

[1]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

Abstract

Machine learning can offer an increase in the flexibility and applicability of robotics at several levels of control. In this paper, we characterize two symbolic learning tasks in the field of robotics. We outline an approach for learning features from sensory data and for using these features to learn more complex ones. We illustrate our approach with first experiments in the field of navigation.

1 Introduction

One of the major cost factors in robot applications is the impossibility of automatically transferring the experience from one application to the next (similar) one. The inflexibility of robot programs makes it a costly process to start a new application. A lot of preparation is necessary before a robot can be utilized successfully. This preparation is not yet well enough supported by systems.

The interaction between the robot and its user is oriented towards the robot's needs. A user-friendly level of interaction has not yet been achieved. Teaching a robot is done using manual guidance techniques or simply by programming (preprogramming). Commanding the robot requires very precise orders. To navigate the robot, the orders refer either to points on the environmental map, or to points relative to the robot's position. Of course, particular objects can be represented by their positions on the map. For this particular environment, the user may use the names of these objects when inputting a navigational task. If, for instance, it is known that a cupboard is in a particular area in a room, an abstract command can be formulated: "move to cupboard". This command can be mapped one-to-one into the move-command with the absolute position. However, in a different environment with a cupboard in another position, the command "move to cupboard" cannot be interpreted. This would require the recognition of cupboards in general, or, in other words, the general concept of a cupboard.

One approach to ease robot applications is to integrate machine learning techniques into robotics. There are three levels of control where learning abilities can be put to good use:

1. the subsymbolic level of *reflexes* where the robot learns to enhance the immediate reaction to sensory input
2. the symbolic level of *concepts* where the robot learns
   - to recognize objects in different environments
   - to enhance a map of the environment
   - which actions are appropriate for easing an object's recognition
3. the *planning level* where the robot learns to enhance a sequence of actions because of experience with prior actions.

We assume a hierarchical robot architecture where several learning techniques are applied at all levels [Knieriemen, 1991]. At all levels, perception-oriented and action-oriented processing is related (see figure 1). Regarding concepts within this framework, the representation of relations between operational concepts is at the highest level, the operational concepts themselves are at the second level, and their execution, i.e. sensing and performing elementary operations, is at the lowest level. *Operational concepts* integrate perception-oriented and action-oriented features. Moreover, the perceptual features are constructed with respect to actions and the actions are described with respect to the perceptual features. The relations between operational concepts are used for planning at a higher level. Concepts are related by sharing features and being discriminated by other features. Features may refer to perceptions or to actions. The most typical situation is that some concepts share perceptual features and are discriminated by an action which leads to other perceptual features. There are some features which trigger the recognition of an object. Most often, these are perceptual features. For instance, a particular sensor pattern indicates that the sensed object could be a cupboard, a table, or a projection on a wall, e.g., a column. Some of the objects can be discriminated by an action. For instance, one can put something under a table but not under the projection on a wall because columns are straight upright from top to bottom. There are actions that distinguish objects. The result of the action needs to be verified again by sensor patterns (e.g., has the object been pushed under the table?). The trigger-action-verification cycle is used for planning on the highest level.

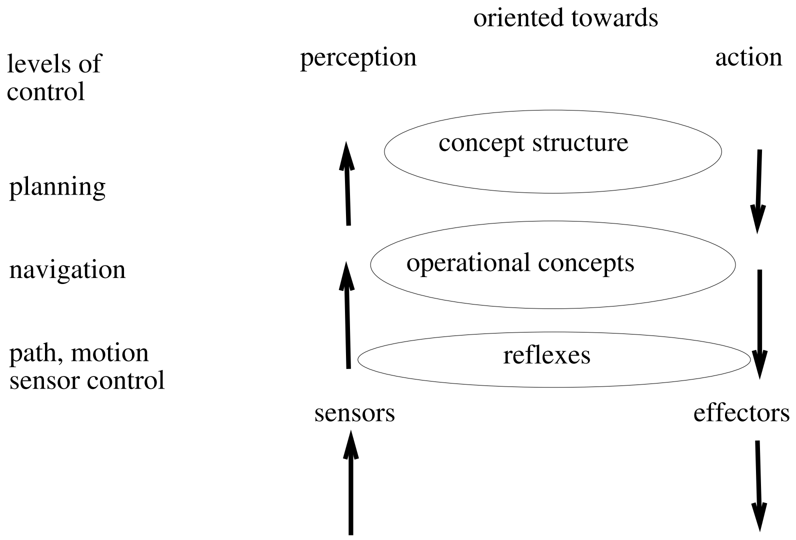

Figure 1: Levels of control

In this paper, we discuss the integration of learning into robotics only with respect to the second level, the symbolic level of concepts. In particular, we discuss the first step, the representation and learning of the action-oriented perceptual features for operational concepts. In section 2 we describe two particular learning tasks and present our view of features for operational concepts. In section 3, we describe how the data, from which we learn, are gathered. Section 4 describes the experimental preprocessing of sensory data and the representation of features and concepts. Section 5 gives an example of learning an operational concept. The conclusion evaluates our approach and relates it with other work.

2 Two learning tasks

In artificial intelligence, concept learning refers to learning the description of a target concept that covers the extension of that concept. The description consists of features and is expressed by a formula which is true or false for a particular object. If the formula is true for an object, the object is a member of the concept, otherwise it is not a member. This view of concept learning needs to be qualified if we want to integrate learning into robotics. Two questions point to the weakness of the classical view of concepts for robotics applications of learning.

The first question is *"Where do the features come from?"*

In most machine learning applications, the features are given. In robot applications, sensory data are given. This is what the robot perceives from the environment. However, sensory data are too specific to apply to more than one object perceived during one operation of the robot. Because of this, features have to be calculated from the sensory data. This feature construction[1] is a difficult task. Depending on the features, concept learning may become more or less efficient. Up to the present, there is no theory of representation that tells us which features to construct. Usually, the application developer programs the construction of a set of features. He then calls up a learning algorithm which learns concept descriptions. The concept description uses some of the constructed features. The application developer then evaluates the quality of the learning result. If the evaluation is not good enough, the developer tries out the next way of calculating features from the sensory data. The process continues until the evaluation produces a satisfactory result. This time-consuming *preprocessing* of data should be supported by a system. We may view it as a learning task in its own right.

Learning basic features Several ways of abstracting basic features from sensory data are prepared and partially ordered. The first variant is selected and concept learning is tried using the abstracted sensory data. The learning result is evaluated. If the evaluation is not good enough, the next variant of abstracting features from data is chosen, and concept learning is called again. This loop goes on forever. If appropriate ways of calculating features are available, the appropriate degree of granularity of the features should be learned.

A similar idea was proposed by Wrobel as an answer to symbol grounding problems [Wrobel, 1991]. In Wrobel's paper, however, learning features from sensory data was modeled as a direct segmentation task of a stream of real-valued sensory input. In contrast, we provide the algorithm with more complex calculations such as, for the gradient of two values measured consecutively, or for the difference of two angles. We do not aim at learning such functions. What we do want to learn is the appropriate degree of granularity of the abstraction result, and that means which function to choose.

The second question which points out deficiencies of the classical view of concepts is *"How do we verify that a particular object is a member of a concept?"*

Most often, if all features defining a concept are true for an object, it is concluded that the object is a member of the concept. This means that a concept is completely determined by its features. Think, for instance, of a representation for the everyday concept "cup". The flat bottom, the concave form, and the handle could be features of the concept "cup", but features alone are not sufficient to define a cup. A particular object could be described by these three features without actually being a cup. DeJong and Mooney have shown the example of a receptacle with a handle bridging over the opening of the concave form

[1] Sometimes, the euphemistic term "feature extraction" is used. It is a misleading term, as the features are not a subset of sensory data.

[De Jong and Mooney, 1986]. You cannot drink from such a receptacle. Therefore, it cannot be a cup. Of course, one can add a feature stating that the handle must be on the side. In an infinite number of ways, however, a given receptacle can be such that it is impossible to drink from it. All these ways cannot be excluded by features in the concept and object descriptions. Presumably, for any list of features that a cup cannot have, we could construct an additional exceptional feature which hinders drinking from a receptacle. This is the frame problem [McCarthy and Hayes, 1969].

The frame problem indicates that observational features alone are not adequate. What is most important about a cup is that one can drink from it. Drinking is a verifying action for a cup. Even a baby cup which does not have a flat bottom (but a ball filled with heavy material so that it stands upright) is a cup because one can use it for drinking. As many ways as there are to disturb the functionality of a cup, there are to preserve its functionality even if the features are not true. A concept description should not only consist of perceptual features but also of a *verifying action*. If the action is successful for a particular object, it belongs to the concept. If the action is not successful, it does not belong to the concept. In this way, actions are integrated into concept descriptions and into their application as recognition functions.

Similarly, Giordana has proposed to use *executable features* (executable predicates) in concept descriptions [Giordana et al., 1990] . These features are true for an object if a particular handling of this object is successful. For instance, the feature "movable" for a concept can be verified by moving that object. We want to go one step further and propose that even the perceptual features should be oriented towards action and action features should be oriented towards perceptions.

The perceptual features describe patterns which are perceived while the robot performs an action. Even features that seem to be purely observational without any link to an action are, in fact, action-oriented. The perception of a flat bottom, for instance, is only possible when looking from a particular angle with respect to the object. Looking straight down upon the receptacle does not allow one to determine whether the bottom is flat or not. A perceptual feature (e.g, flat bottom) is constructed with reference to an action (e.g. looking from the side). Action features, in turn, require perceptual features. Actions are represented in terms of the following sensor patterns: what is sensed before a particular action can be performed, what is sensed during successful performance of the action, and what is sensed as the outcome of the action. Hence, perception and action are closely interrelated at all levels of abstraction.

If we had a robot that could drink, there would be features constructed from sensory data such as `full receptacle` and `empty receptacle`. The operational concept of a `drinking receptacle` would look roughly like this:

```
full_receptacle(Obj, T1, T2, lifting) &
at_lips(Obj, T2, T3, slurping) &
...  &
empty_receptacle(Obj, Tn-1, Tn, lowering)
→ drinking_receptacle(T1, Tn, drinking)
```

Of course, this is an unrealistic example. We describe our experiments in the navigation task in 5.

Learning more abstract features

Given basic features which describe sensory data which were measured during a particular action (result of the first learning task), and given the interval of time in which an object was measured by some sensors, learn higher-level action-oriented perceptual features.

Note, that the higher-level features need not be expressed by just one rule. An inference chain may lead from the lowest feature to the highest one.

Before we describe in detail our experiments in learning action-oriented perceptual features at several levels of abstraction, we want to introduce the navigational task and the data that we have.

3 Data from a navigation scenario

Our learning tasks are embedded in enhancing the flexibility of autonomous vehicle navigation. A hierarchical architecture of the navigation system is assumed [Kaelbling, 1987]. Whereas reinforcement learning is applied to low-level learning tasks such as obstacle avoidance [Millan and Torras, 1991] and learning macro-operators is applied to the planning level [Spandl and Pitschke, 1991] we investigate learning operational concepts from sensory data. The data are gathered by a vehicle while moving through a room. The vehicle, PRIAMOS, has been developed at the University of Karlsruhe. The vehicle is able to turn around 360 degrees at the same point and can move in every direction with every orientation. The vehicle is equipped with 24 sonar sensors, all of which are loccated at the same height all around the vehicle. The aim is to enable the navigation system to perform high-level tasks such as "pass through the door, turn left, move to the cupboard" in various environments. This requires the learning of operational concepts such as "pass through door" from several training traces. Each training trace consists of the following data for each point of time and each sensor:

> trace number, point of time, sensor number, measured distance, sensor orientation, sensor position in the global coordinate system, orientation of the robot, robot position in the global coordinate system, object number and edge number of what is sensed by the distance signal.

Currently, we have 28 traces, each with 27 time points. Therefore, there are 18144 measurements from which we can learn. Most paths are movements through a door with different distances from the doorframes, and with or without a cupboard at the wall close to the door. A room with its edge numbers is shown in figure 2. Most edges represent walls of the room. The edge numbers of the cupboard (0 to 3) are printed in italics. In order to obtain examples for learning, we have grouped together all measurements with the same orientation of the robot and all measurements which sensed the same edge. Particular constellations of edges, such as two walls being linked by a right angle, are also gathered. These constellations are called "concave" and "convex". For instance, edges 7 and 6 in the room of figure 2 form a concave constellation with respect to the robot's position (hatched area). Edge 7 alone is just a "line", be it measured from parallel, straight towards the

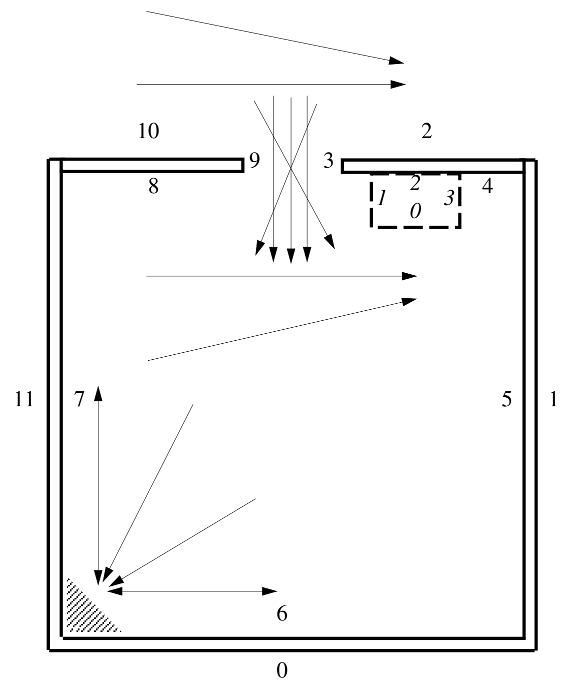

Figure 2: Room with traces and edge numbers

edge, straight away from the edge, or diagonal position. Two parallel edges, e.g. edge 9 and 7 or 3 and *1*, are named "jump". The paths of the robot are indicated by the arrows.

From the traces, we have derived 23 examples for concave edges, 57 examples for convex edges, 206 for jump, and 718 for line.

4 The representation

The representation formalism we use is the one of the system MOBAL[2]. It is a restricted first order logic with explicit negation, only one conclusion, and all arguments of the conclusion occurring in at least one premise. The representation formalism is capable of expressing higher-order constructs[3]. Among other representation items, MOBAL offers:

Facts Facts are used to state relations, properties of objects, and concept membership. Facts are represented as function-free literals without variables. A derived or input fact without explicit negation is interpreted as `true`. Every fact which is to be interpreted as `false` must be explicitly negated. An example for a fact is `s_jump(7,5,11,26,diagonal)`, where `7` is the trace number, `5` is the sensor number, `11` is the starting point, `26` the end point of the time interval, and `diagonal` is the orientation of the robot towards the sensed edge.

[2]MOBAL is developed at the German National Research Center for Computer Science.

[3]Formal properties of the formalism have been proved in [Wrobel, 1987]. A description of the system including detailed chapters about the representation formalism is [Morik et al., 1993].

Rules We may view the rules in MOBAL as Horn clauses. If the premises can be instantiated by positive facts, the conclusion is derived. An example for a rule is the following

```
stable(Trace,SAlpha,S,T1,T2,Value1) &
incr_peak(Trace,SAlpha,S,T2,T3,Value2) &
stable(Trace,SAlpha,S,T3,T4,Value3)
→ s_jump(Trace,S,T1,T4,parallel).
```

This rule expresses a sequence of sensor patterns. First, in the time interval from `T1` to `T2`, about the same gradient holds for any subsequent measurements. From `T2` to `T3`, there is a much higher gradient, and the higher gradient is stable from `T3` to `T4`. This sequence of features concludes in a more abstract feature, namely `s_jump`.

Rule schemata Rule schemata express the structure of rules to be learned. They provide the user of MOBAL with an explicit control over the hypothesis space for learning. A rule schema is a rule in which predicate variables are used instead of actual predicates of the application domain. A predicate variable can be instantiated by a predicate symbol of the same arity. There is a substitution Σ for predicate variables. Let RS be a rule schema, then RSΣ is an (partially) instantiated one. If all predicate variables in RS are substituted by predicate symbols, RSΣ is a rule.

```
P1(Trace,SAlpha,S,T1,T2,Value1) &
P2(Trace,SAlpha,S,T2,T3,Value2) &
P1(Trace,SAlpha,S,T3,T4,Value3)
→ P3(Trace,S,T1,T4,M).
```

According to Kietz and Wroble [Kietz and Wrobel, 1991], rule schemata are ordered with respect to their generality such that the generality of fully instantiated rule schemata is given by theta-subsumption [Plotkin, 1970].

Within the representation formalism, the representation language for our application has to be declared. MOBAL supports the declaration of predicates or acquires the declarations automatically. We have developed a hierarchy of predicates. The lowest level is given by the `measure` predicate. It expresses a sensor measurement as a logical fact in the following form:

```
measure(Tr, Ti, S, Dist, SAlpha, SX, SY, SZ, PX, PY, PZ, Oi, Ei)
```

where

`Tr`: trace number,
`Ti`: point of time, for instance, T1 and T2 are points of time,
`S`: sensor number,
`Dist`: measured distance,
`SAlpha`: sensor orientation,
`SX,SY,SZ`: sensor position in the global coordinate system,
`PX,PY,PZ`: position of the sensed point in the global coordinate system,
`Oi`: object number, and
`Ei`: edge number of what is sensed by the distance signal.

The next higher level is given by the following predicates which express sensor patterns: `no_measurement`, `increasing`, `incr_peak`, `decreasing`, `decr_peak`, `single_peak`, `stable`, `straight_to`, `straight_away`. They all have the arguments, `Tr`, `SAlpha`, `S`, `T1`, `T2`, `Grad`, where `Grad` is the calculated value of the chosen function for abstracting data and all other abbreviations are as above. It need not be the gradient, but right now, the function is the gradience. Several gradients are abstracted into one qualitative predicate by a tolerance function. The first learning task, introduced in section 2, is about adjusting these functions to get the proper calculation of these sensor patterns.

From these sensor patterns, our basic features, more abstract features are learned. There is an inferential hierarchy of features. The basic features occur in the premises of rules where the conclusion is a more abstract feature. These more abstract features occur in premises of rules where the conclusion is an even more abstract feature. This inferential hierarchy can be viewed as a hierarchy of features, but it can as well be considered a hierarchy of simple concepts. Because we do not use propositional logic, no principle distinction need be made between features and concepts. The interesting point is that from the lowest inferential level on sensor patterns are defined with respect to robot actions, and this interrelation is continued at all inferential levels.

Predicates concluded from sensor patterns of one sensor are: `s_jump`, `s_line`, `s_convex`, `s_concave`. The predicate arguments are `Tr`, `S`, `T1`, `T2`, `M`, where `M` means one of the following movements (relative orientation): `diagonal`, `parallel`, `straight_away`, `straight_towards`. All other abbreviations are as above. These groupings were chosen to represent the next level of abstraction because on one hand they can be derived from a geometric description of the map defined with respect to a global coordinate system. On the other hand they can be perceived by the robot yielding a description defined with respect to the robot's reference system. Thus they support the transformation from robot reference system to the global one and vice versa.

The relative orientation of the robot with respect to a group of edges is defined identically for s_line and s_jump: If the direction of the robot's movement is parallel to the edge(s) it is labeled parallel. If the angle between the robot's direction of movement and the edge orientation is a right angle it is labeled straight_away or straight_towards. All other cases are labeled diagonal. For convex and concave corners, respectively, the relative orientation is considered parallel if the robot orientation is parallel to one of the edges of the corner. The relative orientation is called straight_towards or straight_away if the robot's orientation is not parallel to each of the two edges and it moves towards or away from the point, which both edges share. All other cases are labeled diagonal. From these predicates one can conclude more general features for a group of sensors: `sg_jump`, `sg_line`, `sg_convex`, `sg_concave` with the same arguments as their corresponding features for one sensor. In this way it can be expressed that all sensors at one side of the robot perceive the same sensor pattern.

From these features we can conclude simple operational concepts, such as `move_through_door(Tr, T1, T2, M)`. As the meanings of the features or concepts are learned, the rules are described in section 5.

5 Learning what it means to pass through a doorway

Referring to figure 2 we illustrate what happens when the robot goes through a doorway [4]. Consider the traces where the robot moves through the doorway parallel to the doorframes. When approaching the door, the sensors at the front right corner of the robot perceive the convex corner produced by edges 10 and 9. The sensors on the right side perceive the jump caused by edges 9 and 7. Then the sensors on the right back corner perceive the convex corner caused by the pair of edges 9 and 8. Correspondingly, the sensors on the front left corner first perceive the convex corner caused by edges 2 and 3, and then the concave corner produced by edges 5 and 6. Meanwhile, the sensors on the left side perceive the jump caused by edges 3 and 1, and the jump caused by edges 1 and 5. The sensors at the front constantly measure the back wall of the room, edge 6. This is what human inspectors of the sensory data realize. The issue now is to have the system detect these relations by machine learning.

The learning module of the MOBAL system, RDT [Kietz and Wrobel, 1991], is applied to solve this complex learning task. RDT is a model-based inductive learning algorithm which learns instantiations of rule schemata. RDT learns from most general rules to more special rules. In contrast to learning algorithms that learn most specific generalizations, RDT learns most general rules that obey the user given evaluation criterion. In contrast to systems that stop after having found a good rule, RDT learns as many most general rules as possible[5].

In this application, RDT learns the following three types of rules:

- Rules for patterns for single sensors (see section 5.1)
- Rules for patterns for groups of sensors (see section 5.2)
- Rules for going through a door in terms of patterns for sensor groups (see section 5.3)

For each type of rule, a set of rule schemata is prepared. With the sensor data prepared as described in section 3, rules of all types are learned. We illustrate the learning step by step in the next sections. Note, however, that MOBAL is able to learn many rules and rules at different inferential levels in the same run.

5.1 Learning rules for single sensors

To illustrate the first type of rules, we consider how a single sensor perceives a jump, i.e. two parallel edges at different distances relative to the robot. The following literals represent the features perceived in trace `19` by sensor `5` during time interval `[9,26]`: `stable(19,180,5,9,10,-1)`, `incr_peak(19,180,5,10,11,85)`, `stable(19,180,5,11,26,0)`. Sensor `5` is located at the right side of the robot. Trace `19`

[4] In the following we use going through a door as synonym for going through a door way. Only when talking with a native speaker of the American language we realized that the German concept of "door" which has as a crucial point of its meaning that one can pass through a door does not correspond to the American concept of a "door" which has as part of its meaning that one cannot pass through it. We define the German concept of a door in this paper.

[5] Of course, RDT does not further specialize rules which have been accepted already.

is one of the traces where the robot moves through the door parallel to the doorframes (see figure 2). In the preprocessing step, we calculated the fact `s_jump(19,5,9,26,parallel)`. This literal represents the fact that in trace `19`, sensor `5` perceived a jump during time interval `[9,26]`, when moving parallel to the two edges. This fact was derived using the knowledge about pairs of edges which produce an `s_jump`. In addition we used knowledge about edges which were measured by certain sensors during certain time intervalls. For the cases where the robot moves diagonally through the door, e.g. traces `7` and `8`, we have the features `incr_peak(7,213,5,11,12,86)`, `increasing(7,213,5,12,26,29)` and `incr_peak(8,326,5,11,12,86)`, `increasing(8,326,5,12,26,29)`. The corresponding jump-predicates are `s_jump(7,5,11,26,diagonal)` and `s_jump(8,5,11,26,diagonal)`.

The other sensors on the right side of the robot, which are located behind sensor `5`, perceive the same features at successive time points. Overall 206 examples for s_jump are available for RDT. In addition, RDT is given 30 rule schemata. Two of them are listed here for illustration:

```
P1(Trace, Orientation, Sensor, Time1, Time2, Grad1) &
P2(Trace, Orientation, Sensor, Time2, Time3, Grad2)
→ Q(Trace, Sensor, Time1, Time3, parallel).
```

and

```
P1(Trace, Orientation, Sensor, Time1, Time2, Grad1) &
P2(Trace, Orientation, Sensor, Time2, Time3, Grad2) &
P1(Trace, Orientation, Sensor, Time3, Time4, Grad3)
→ Q(Trace, Sensor, Time1, Time4, diagonal).
```

In the rule schemata capital letters refer to variables (predicate variables and argument variables), and `parallel` and `diagonal` refer to constants. The rule schema can only be instantiated such that the last argument of the concluding predicate has the constant as its last argument. In this way, the hypothesis space for learning is further focused on our learning goal. 28 rules were learned for s_jump, for instance:

```
incr_peak(Trace, Orientation, Sensor, Time1, Time2, Grad1) &
increasing(trace, Orientation, Sensor, Time2, Time3, Grad2)
→ s_jump(Trace, Sensor, Time1, Time3, diagonal).
```

and

```
stable(Trace, Orientation, Sensor, Time1, Time2, Grad1) &
incr_peak(Trace, Orientation, Sensor, Time2, Time3, Grad2) &
stable(Trace, Orientation, Sensor, Time3, Time4, Grad3)
→ s_jump(Trace, Sensor, Time1, Time4, parallel).
```

In tables 1 and 2 we have summarized the results of learning rules for patterns for single sensors. Table 1 shows the number of examples for each grouping and relative orientation of the robot towards the grouping. Table 2 shows the number of learned rules for each grouping and relative orientation.

5.2 Learning rules for groups of sensors

In the next step, we tried to learn rules which define patterns for groups of sensors. The following ideas motivated our approach. The fact that a group of sensors belonging to the same class (e.g. sensors on the right side of the robot) perceived the same pattern yields more evidence than the fact that only a single sensor perceived the same pattern.

	s_line	s_jump	s_convex	s_concave
parallel	200	103	44	16
diagonal	383	89	13	4
straight_towards	75	6	0	1
straight_away	60	8	0	2
total	718	206	57	23

Table 1: Number of examples for patterns for single sensors

	s_line	s_jump	s_convex	s_concave
parallel	14	11	16	3
diagonal	25	15	5	3
straight_towards	16	2	0	1
straight_away	13	0	0	2
total	58	28	21	9

Table 2: Number of learned rules for patterns for single sensors

In addition we get the information to which class a sensor belongs. In this way we get a representation independent of particular sensor numbers.

In each of our example traces for moving through a door, all the sensors on the right side of the robot perceive the jump caused by edges 9 and 8 at successive time points. This is expressed by the three groups of literals:
`s_jump(7,5,11,26,diagonal)`, `s_jump(7,6,12,26,diagonal)`,
`s_jump(7,7,13,26,diagonal)` and
`s_jump(8,5,11,26,diagonal)`, `s_jump(8,6,12,26,diagonal)`,
`s_jump(8,7,13,26,diagonal)` and
`s_jump(19,5,9,26,parallel)`,`s_jump(19,6,10,26,parallel)`,
`s_jump(19,7,11,26,parallel)`.

The fact that these sensors belong to the class "right_side" is expressed with the predicate `sclass(<trace>,<sensor>,<tmin>,<tmax>,<sensor_class>)`, where `<tmin>` and `<tmax>` denote the time interval for which the membership is valid. For trace 7 and sensors 5, 6, and 7 we have `sclass(7,5,1,26,right_side)`, `sclass(7,6,1,26,right_side)`, and `sclass(7,7,1,26,right_side)`. To express the fact that a time point is the direct successor of another time point we use the predicate `succ(<time>,<time>)`.

In this learning step, we use rule schemata which focus the search in the hypothesis space of RDT on hypotheses which constrain the required number of sensors of a specific class. These sensors have to perceive the same pattern. We consider groups with one, two, and three sensors. Via the rule schemata, we also put constraints on the time intervals during which the same pattern has to be perceived by the sensors. The sensors in the grouping have to perceive the same pattern in the same time interval or in time intervals with successive starting or ending points. Examples for rule schemata are:

```
S_Pattern(Trace,Sensor,Start,End,Movement) &
sclass(Trace,Sensor,Time1,Time2,Class) &
Const_Class(Class) & Const_Move(Movement) &
Time1 ≤ Start & End ≤ Time2
→ SG_Pattern(Trace,Class,Start,End,Movement).
```

and

```
S_Pattern(Trace,Sensor1,Start,End,Movement) &
S_Pattern(Trace,Sensor2,Start,End,Movement) &
sclass(Trace,Sensor1,T1,T2,Class) & T1 ≤ Start &
sclass(Trace,Sensor2,T1,T2,Class) & End ≤ T2 &
Const_Class(Class) & Const_Move(Movement) &
Sensor1 ≠ Sensor2
→ SG_Pattern(Trace,Class,Start,End,Movement).
```

and

```
S_Pattern(Trace,Sensor1,Start1,End1,Movement) &
S_Pattern(Trace,Sensor2,Start2,End2,Movement) &
S_Pattern(Trace,Sensor3,Start3,End3,Movement) &
succ(Start1,Start2) & succ(Start2,Start3) &
sclass(Trace,Sensor1,T1,T2,Class) & T1 ≤ Start1 &
sclass(Trace,Sensor2,T1,T2,Class) & End3 ≤ T2
sclass(Trace,Sensor3,T1,T2,Class) &
Const_Class(Class) & Const_Move(Movement) &
→ SG_Pattern(Trace,Class,Start1,End3,Movement).
```

The question is why we have included rule schemata for sensor "groups" that have one sensor. In this way we have found that convex and concave corners are almost always perceived by a single sensor positioned at a corner of the robot. In this case the rules determine the class of the sensor. An example for a learned rule for a group with one sensor is

```
s_concave(Trace,Sensor,Start,End,Movement) &
sclass(Trace,Sensor,T1,T2,Class) &
corner_back_left(Class) & diagonal(Movement) &
T1 ≤ Start & End ≤ T2
→ sg_concave(Trace,Class,Start,End,Movement).
```

This rule tells us, that a sensor on the back left corner of the robot perceives a concave corner when the robot moves diagonally along this concave corner. The following rules show more examples for those rules that have been learned:

```
s_line(Trace,Sensor1,Start,End,Movement) &
s_line(Trace,Sensor2,Start,End,Movement) &
sclass(Trace,Sensor1,T1,T2,Class) &
sclass(Trace,Sensor2,T1,T2,Class) &
front_middle(Class) & straight_towards(Movement) &
T1 ≤ Start & End ≤ T2 &
Sensor1 ≠ Sensor2
→ sg_line(Trace,Class,Start,End,Movement).
```

and

```
s_jump(Trace,Sensor1,Start1,End1,Movement) &
s_jump(Trace,Sensor2,Start2,End2,Movement) &
s_jump(Trace,Sensor3,Start3,End3,Movement) &
succ(Start1,Start2) & succ(Start2,Start3) &
sclass(Trace,Sensor1,T1,T2,Class) &
sclass(Trace,Sensor2,T1,T2,Class) &
sclass(Trace,Sensor3,T1,T2,Class) &
right_side(Class) & parallel(Movement) &
T1 ≤ Start1 & End3 ≤ T2
→ sg_jump(Trace,Class,Start1,End3,Movement).
```

The former rule tells us that the sensors on the front of the robot perceive a line during the same time interval, when the robot moves straight towards an edge. The latter rule tells us that if the robot moves parallel along a jump on its right side, three sensor on the right side perceive the jump in time intervals whose starting points follow each other.

These kind of rules enable the robot to focus its attention on specific sensors in order to detect a certain grouping and to gather evidence for it.

5.3 Learning the meaning of moving through a doorway

In the last step, we learn rules for going through a door by using patterns for sensor groups. In the beginning of this section we described which groupings of edges (jumps, concave and convex corners, lines) the robot perceives while moving parallelly through a doorway. The question to be asked is which subset of these features is sufficient to discriminate between going through a doorway on one hand, and on the other hand, passing by a door, or moving towards a corner of the room, etc.

We have used rule schemata which reflected the symmetry of the doorframes, which could be detected by classes of sensors opposite to each other. By this we mean the classes "left_side" and "right_side", for example. The sensors in these classes perceive the jump caused by edges 9 and 7 on the right side and the jump caused by edges 3 and 1 (3 and 5), respectively. An example for a rule schema is

```
SG_Pattern(Trace,right_side,T1,T2,Movement) &
SG_Pattern(Trace,left_side,T1,T2,Movement) &
Const_Move(Movement) & Start ≤ T1 & T2 ≤ End
→ Q(Trace,Start,End,Movement).
```

In this case, the constants `right_side` and `left_side` focus RDT's search on rules associated with these sensor classes.

Depending on the orientation of the robot, with respect to the doorframes, the time intervals for the left and right side are either the same or have a short time delay. This is reflected in the learned rules for moving parallelly through a door versus moving diagonally through a door. The learned rules for moving parallelly through a door are:

```
sg_jump(Trace,right_side,T1,T2,Movement) &
sg_jump(Trace,left_side,T1,T2,Movement) &
parallel(Movement) & Start ≤ T1 & T2 ≤ End
→ move_through_door(Trace,Start,End,Movement).
```

and

	Number of examples	Number of learned rules
parallel	6	2
diagonal	4	2
total	10	4

Table 3: Number of examples and rules for moving through a doorway

```
sg_jump(Trace,right_side,T1,T2,Movement) &
sg_jump(Trace,left_side,T3,T4,Movement) &
parallel(Movement) & succ(T1,T3) & Start ≤ T1 & T2 ≤ End
→ move_through_door(Trace,Start,End,Movement).
```

The rules for moving diagonally through a door are the following

```
sg_jump(Trace,right_side,T1,T2,Movement) &
sg_jump(Trace,left_side,T3,T4,Movement) &
diagonal(Movement) & succ3(T1,T3) & Start ≤ T1 & T4 ≤ End
→ move_through_door(Trace,Start,End,Movement).
```

and

```
sg_jump(Trace,right_side,T1,T2,Movement) &
sg_jump(Trace,left_side,T3,T4,Movement) &
diagonal(Movement) & succ3(T3,T1) & Start ≤ T1 & T4 ≤ End
→ move_through_door(Trace,Start,End,Movement),
```

where succ3(<t1>,<t2>) expresses the fact that the time difference between t1 and the following time point t2 is 3. The results of this learning step are summarized in table 3.

6 Conclusion

In this paper we have described one step towards automatically constructing higher and hence more user-friendly notions for human-robot interaction. In contrast to previous approaches to learning in robotics, such as Explanation Based Learning [Segre, 1988], [Zercher, 1992], or subsymbolic techniques [Millan and Torras, 1991], we apply inductive logic programming to robot navigation. Bratko and his colleagues apply inductive logic programming to descriptions of real-world processes, too [Bratko et al., 1992]. Our aim of automatically building up higher and more qualitative levels of describing events is similar to their qualitative modeling of physical systems in very general ways only. Whereas they have used about ten examples and a fixed background knowledge in order to learn one clause that describes a physical system, we constructed many times more examples based on real sensory data in order to learn a hierarchy of action-oriented features. Each learned feature is represented by several clauses. Our application of RDT differs from the heuristic relational learner FOIL [Quinlan, 1990] in that RDT is capable of learning many rules in one run. FOIL selects the best covering and discriminating rule. In our navigation application, however, we want as many rules as possible that could match a new situation. The training phase in which we know the sensed edges should yield rules that can be used in the performance phase, where this knowledge is not available.

Our main results are:

- identification of learning tasks in a framework for learning in navigation;
- representation of action-oriented perceptual features at several levels of abstraction;
- learning action-oriented perceptual features at different levels of abstraction.

Further work is needed concerning the two learning tasks identified in this paper. We chose one way of constructing the basic features (stable, increasing, etc.) from the sensory data. These features are used in the first learning step (cf. 5.1). This learning step resulted in two rules for deriving the same sensor pattern, one rule being a more detailed version of the other one:

```
stable(Trace, Orientation, Sensor, Time1, Time2, Grad1) &
incr_peak(Trace, Orientation, Sensor, Time2, Time3, Grad2) &
stable(Trace, Orientation, Sensor, Time3, Time4, Grad3) &
→ s_jump(Trace, Sensor, Time1, Time4, parallel).
```

and

```
stable(Trace, Orientation, Sensor, Time1, Time2, Grad1) &
incr_peak(Trace, Orientation, Sensor, Time2, Time3, Grad2) &
incr_peak(Trace, Orientation, Sensor, Time3, Time4, Grad3) &
incr_peak(Trace, Orientation, Sensor, Time4, Time5, Grad4) &
stable(Trace, Orientation, Sensor, Time5, Time6, Grad5) &
→ s_jump(Trace, Sensor, Time1, Time6, parallel).
```

The first learning task should adjust the tolerance function for gradience such that the gradients `Grad2, Grad3, Grad4` are considered to be the same and `incr_peak` no longer appears three times. Some ambiguities in the data can be handled in this way.

Concerning the second learning task, a complete and systematic exploration of action-oriented perceptual features can be undertaken now that we have shown the feasibility of our approach.

An interesting research issue is to define perception-oriented action features corresponding to our action-oriented perceptual features. Both types of features will then be integrated into operational concepts. The structure of concepts then require further investigation. Look for instance, at the following four rules for the concepts `c1` or `c2`:

```
r1) p1(X,Y1) & p2(Y1,Y2) & p3(Y2,Y3) → c1(X,Y1,Y3)
r2) p1(X,Y1) & p2(Y1,Y2) → c1(X,Y1,Y2)
r3) p1(X,Y1) & p4(Y1,Y2) & p3(Y2,Y3) → c1(X,Y1,Y3)
r4) p1(X,Y1) & p2(Y1,Y2) & p3(Y2,Y3) → c2(X,Y1,Y3)
```

where the third rule represents the case of a missing feature, i.e. `p4` expresses `no_measurement`. The premises of r2 are a subset of those of r1. Should the subset relation between premise sets indicate a subset relation between concepts? Or, should the rule with more premises result in a more evidence for the concept membership of `X` to `c1`? Or should each premise be interpreted as referring to a point in time where r1 covers the whole time interval, and r2 refers to a part of the interval? Another interpretation of partial information is associated with r3. There, the sensors failed to measure feature `p2`. In addition to the problems with partial information, we have to deal with conflicting information. The disjoint concepts `c1` and `c2` are derived from the same premises. This may indicate that a distinguishing feature is missing. If, however, there is no discriminating feature, and it cannot even be constructed during the course of our first learning task, then a probablistic approach may help to decide what to do. The formal analysis of such problems is the subject of current ongoing investigation.

References

[Bratko et al., 1992] Bratko, I., Muggleton, S., and Varsek, A. (1992). Learning qualitative models of dynamic systems. In Muggleton, S., editor, *Inductive Logic Programming*, chapter 22, pages 437 – 452. Academic Press.

[De Jong and Mooney, 1986] De Jong, G. and Mooney, R. (1986). Explanation-based-learning: A alternative view. *Machine Learning*, 2(1):145–176.

[Giordana et al., 1990] Giordana, A., Roverso, D., and Saitta, L. (1990). Abstraction - a framework for learning. In *Procs. AAAI-Workshop on Automatic Generation of Approximations and Abstractions.*

[Kaelbling, 1987] Kaelbling, L. (1987). An architecture for intelligent reactive systems. Technical report, CSLI, Stanford, Ca.

[Kietz and Wrobel, 1991] Kietz, J.-U. and Wrobel, S. (1991). Controlling the complexity of learning in logic through syntactic and task-oriented models. In Muggleton, S., editor, *Inductive Logic Programming*, chapter 16, pages 335 – 360. Academic Press, London. Also available as Arbeitspapiere der GMD No. 503, 1991.

[Knieriemen, 1991] Knieriemen, T. (1991). *Autonome Mobile Roboter - Sensordateninterpretation und Weltmodellierung zur Navigation in unbekannter Umgebung.* BI Wissenschaftsverlag, Mannheim.

[McCarthy and Hayes, 1969] McCarthy, J. and Hayes, P. (1969). Some philosophical problems from the standpoint of artificial intelligence. In Michie, D. and Meltzer, B., editors, *Machine Intelligence*, volume 4. Edinburgh University Press.

[Millan and Torras, 1991] Millan, J. and Torras, C. (1991). Learning to avoid obstacles through reinforcement. In Birnbaum, L., editor, *Machine Learning - Procs. of the 8th International Workshop*, pages 298 – 302. Morgan Kaufmann.

[Morik et al., 1993] Morik, K., Wrobel, S., Kietz, J.-U., and Emde, W. (1993). *Knowledge Acquisition and Machine Learning - Theory, Methods, and Applications.* Academic Press, London. to appear.

[Plotkin, 1970] Plotkin, G. D. (1970). A note on inductive generalization. In Meltzer, B. and Michie, D., editors, *Machine Intelligence*, chapter 8, pages 153–163. American Elsevier.

[Quinlan, 1990] Quinlan, J. (1990). Learning logical definitions from relations. *Machine Learning*, 5(3):239 – 266.

[Segre, 1988] Segre, A. (1988). *Machine Learning of Robot Assembly Plans.* Kluwer, Boston.

[Spandl and Pitschke, 1991] Spandl, H. and Pitschke, K. (1991). Lernen von Makro-Trajektorien für einen autonomen Roboter. *KI*, pages 12 – 16.

[Wrobel, 1987] Wrobel, S. (1987). Higher-order concepts in a tractable knowledge representation. In Morik, K., editor, *GWAI-87 11th German Workshop on Artificial Intelligence*, pages 129 – 138, Berlin, New York, Tokyo. Springer Verlag.

[Wrobel, 1991] Wrobel, S. (1991). Towards a model of grounded concept formation. In *Proc. 12th International Joint Conference on Artificial Intelligence*, pages 712 – 719, Los Altos, CA. Morgan Kaufman.

[Zercher, 1992] Zercher, K. (1992). *Wissensintensives Lernen für zeitkritische technische Diagnoseaufgaben.* infix, Sankt Augustin.