Association Rules for Learning Behavioral Mappings in Robotics

Thomas Hellström
UMINF 03.12
ISSN-0348-0542
Department of Computing Science
Umeå University
SE-901 87 Umeå, Sweden
thomash@cs.umu.se

September, 2003

Abstract

Intelligent robotics presents a number of interesting challenges for machine learning. In this paper we look at the specific task of finding a mapping from stimuli to response for robots controlled by reactive behaviors. The mapping will be represented by association rules which is a technique commonly used in data mining but so far not very common in robotics. The approach for learning the association rules is to record training data for a manually programmed controller. The data is then used to generate a set of association rules that replaces the manually programmed controller and manages to reproduce the demonstrated behavior. Techniques to deal with overlapping rules and no firing rules are suggested and evaluated. The method is demonstrated with two examples: One simple wall follower behavior and one more complex road sign problem with a mix of two wall following behaviors. The results show that association rules are a very powerful and practical way to implement rule based controllers for reactive robots.

1 Introduction

The field of Intelligent Robotics presents a number of interesting challenges for machine learning. Learning problems can be formulated for a range of tasks important for an intelligent robot. In this paper we specifically look at the task of finding a mapping from stimuli to response for robots controlled by reactive behaviors. Commonly used techniques in this context are fuzzy logic [20, 3, 9], neural networks [13, 25], genetic algorithms [15, 14, 18] and reinforcement learning [16].

The approach presented in this paper is based on association rules [1]. Association rules have been successfully used for data mining where the goal is to explore complex data bases to find patterns that might be useful for various purposes. Typical areas of application are market basket analysis [1, 4], direct marketing [19], intrusion detection [11] and also scientific applications such as analysis of image data [17] and genome analysis [2]. However, association rules have, to the author's best knowledge, so far not been used in published research in learning robotics.

In this paper, the mapping from stimuli to response is represented by association rules, which are introduced in Section 2. The approach for learning the association rules is to record training data by observing a manually programmed controller that runs the robot for simple tasks like wall following. The training data is then used to generate a set of association rules that replaces the manually programmed controller. The general method is described in Section 3. Results of practical experiments are presented in Section 4, including a simple wall following task and a more complex road sign following behavior. Techniques to deal with the situation when no rules fire are discussed in Section 5. Section 6 contains suggestions for future research and Section 7 concludes the paper with a summary and conclusions.

2 Association Rules

Association rules is a way of expressing dependencies between items in databases [1]. Association rules have the general form $X \Rightarrow Y$, where both X and Y are sets of items. Given transactions $T \in D$, where D is a database and each transaction is a set of items, the rule $X \Rightarrow Y$ expresses a statistical correlation between X and Y. The rules can be constructed according to different quality measures for different purposes. The definitions below follow the notation in [23].

The *coverage* of the rule $X \Rightarrow Y$ is defined as

$$coverage(X \Rightarrow Y) = cover(X),$$

where $cover(X)$ is defined as the number of transactions containing all items in X, divided by the size of the database. I.e., the *coverage* is the fraction of transactions in the database that contain all items in the left hand side X of the rule.

The *support* measures the fraction of transactions that contain all items in both X and Y:

$$support(X \Rightarrow Y) = cover(X \cup Y).$$

For some applications, the statistical correctness of the correlation is critical. The important measure for this quality is called *strength*. The *strength* (sometimes also called *confidence*) of an association rule $X \Rightarrow Y$ is the proportion of the transactions that contain X that also contain Y. It can be computed as

$$strength(X \Rightarrow Y) = \frac{support(X \Rightarrow Y)}{coverage(X \Rightarrow Y)}.$$

Coverage and *support* are of interest when estimating the significance of the *strength,* since they quantify how many observations of X and Y the computation of *strength* is based on.

Some applications are concerned with finding correlations that reflect an underlying dependency between the measured, often physical, entities, i.e. not only a statistical correlation. For example, a rule $LeftWallClose \Rightarrow NoFoodInSight$ may have *strength* 0.95, meaning that the robot can not see any food in 95% of all cases where it is close to the left wall. However, such a rule is unlikely to be of interest if the robot only sees food in 5% of all cases. In such case the concepts *lift* and *leverage* are useful. The *lift* is defined as

$$lift(X \Rightarrow Y) = \frac{support(X \Rightarrow Y)}{cover(X) \cdot cover(Y)},$$

which is the ratio of the joint frequency of X and Y and the frequency that would be expected if the two were independent. Values significantly higher or lower than 1 indicate a dependence between X and Y. One problem with this definition is that a rule with a high *lift* still may be of little interest because it appears very infrequently. The concept *leverage*, defined as

$$leverage(X \Rightarrow Y) = support(X \Rightarrow Y) - cover(X) \cdot cover(Y)$$

is therefore often preferred since it measures the number of appearing transactions above or beyond those that should be expected if one assumes independence.

In general, the process of finding association rules has to deal with two problems [10]. First, Algorithmic complexity. The number of possible rules grows exponentially with the number of items in the database. Modern algorithms have been constructed to efficiently prune the search space based on given thresholds for quality measures on the rules. The second problem is to know which of the generated rules, often many thousands, one should use once they are generated. Also for this problem, different quality measures are useful.

In this paper, all experiments utilize a standard software Magnum Opus [8] based on the OPUS algorithm [21] to generate the association rules. A survey and comparison of algorithms for mining association rules can be found in [10].

3 Formulating the Learning Problem

In the experiments in this paper, control of the robot is purely reactive and is defined by a control law $B : S(t) \rightarrow R(t)$, where S is the vector of stimuli available at time t (a purely reactive scheme involves only stimuli from the current time t,) and $R(t)$ is the response vector issued at time t. B will be implemented as a rule base of rules of the form $S \Rightarrow R$. S is a conjunction of boolean expressions $s_i = v_i$, where s_i is a discretized sensor variable or derived expressions thereof and v_i is an integer value. R has the form $y = a$ where y is a discretized response variable and a is an integer value. With this notation, a rule has the general form

$$s_i = v_i \wedge s_j = v_j ... \wedge s_k = v_k \Rightarrow y = a. \tag{1}$$

In our experiment we are using a Khepera robot (Figure 1) with 8 infrared sensors $IR_0, IR_1, .., IR_7$ to measure distance to the closest obstacle. Each sensor delivers an integer between 0 (correspond-

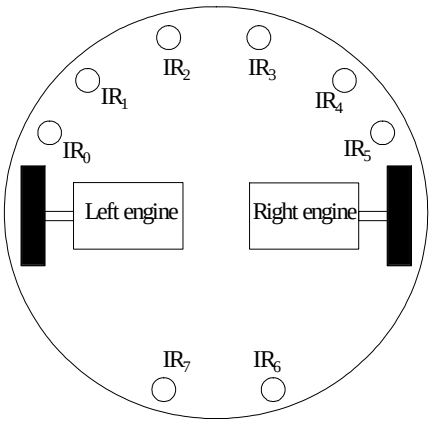

Figure 1: The Khepera robot used in all experiments. The infrared sensors react on reflections from walls closer then approximately 50mm. The two engines control speed and turning.

ing to a distance larger than the sensor range which is about 4 cm.) and 1023 (corresponding to a distance less than about 1 cm.). Each sensor readout IR_i is split into 3 ranges $0, 1, 2$ and represented by a discrete variable ir_i according to

$$ir_i = \begin{cases} 0 \text{ if } & 0 \leq IR_i < 300 & \text{(long distance)} \\ 1 \text{ if } & 300 \leq IR_i < 900 & \text{(medium distance)} \\ 2 \text{ if } & 900 \leq IR_i \leq 1023 & \text{(short distance)} \end{cases} . \tag{2}$$

The robot has two wheels with independent motor control such that both robot speed and turning radius are controlled by setting the left and right speed values v_l and v_r. v_l and v_r can be set to integer values in the range [-127,127]. The response y in our experiments is a coded combination of v_l and v_r according to:

y	v_l	v_r	*Action*
3	-5	5	anti clockwise on the spot
2	0	5	anti clockwise around left wheel
1	2	5	soft anti clockwise
0	5	5	straight ahead
-1	5	2	soft clockwise
-2	5	0	clockwise around right wheel
-3	5	-5	clockwise on the spot

(3)

As an example, a rule for a left-wall-following behavior may look like this:

$$ir_1 = 0 \wedge ir_2 = 1 \Rightarrow y = 1.$$

The rule should be interpreted as follows:

$$\text{if } 0 \leq IR_1 < 300 \wedge 300 \leq IR_2 < 900 \text{ then } v_l = 2 \text{ and } v_r = 5.$$

In plain English this reads as:

if IR_1 senses a long distance and IR_2 senses a medium distance then turn soft anti clockwise.

3.1 Extracting Rules from Training Data

The rule base to control the robot will be generated from training data obtained by a manually programmed controller. A related, somewhat more real-world approach, is to remote control the robot to perform the required task. In either way we obtain a set of stimuli/response pairs that can be used to automatically generate a rule base. This rule base will then replace the controller and hopefully produce the same behavior as the manually programmed controller. Each sample has the form

$$ir_0, ir_1, ..., ir_7, y \tag{4}$$

where each ir_i is an infrared sensor readout and y is the commanded velocity signals from the manually programmed controller. The rules we want to find have the form defined in (1), where each term is an attribute-value pair of the form $s = v$, where s is a discretized sensor variable and v is an integer value. Attribute-value pairs can replace the item concept in the association rule framework described in Section 2, and the rules (1) can be viewed as a restricted form of all possible association rules for the items in (4). By restricting the right hand side of the rules to contain only the y variable, the generated rules can be used for control purposes since they express the mapping $S \rightarrow R$.

Algorithms that efficiently search large databases for association rules have been previously developed [1, 10]. We utilize a standard software Magnum Opus, based on the OPUS algorithm, to generate the association rules. Like most algorithms for mining association rules, OPUS performs a tree based search through the space of all possible association rules with the given left and right hand side terms. During search, the algorithm can automatically filter out rules that are likely to be of little interest. An association rule is said to be *insignificant* if there is another rule with the same right hand side, and a subset of the left hand side for which the *strength* of the former rule is not significantly higher that the *strength* of the latter. A binomial test is used to test for significance. With our settings, a rule is rejected if the probability of obtaining the observed *coverage* and *support* by chance would be less than 0.05 if the data were a random sample and the association's true *strength* were that of the more general rule. For more details refer to [22].

The generated association rules are output along with the quality measures *coverage*, *support*, *strength, lift* and *leverage* (see Section 2 for definitions). In our experiments, the rules are sorted by *strength* and the rules with the highest *strength* are selected for the rule base in the controller. However, for reference and completeness, all tables include all five quality measures.

3.2 Building a Controller

The generated rules are implemented as a controller in the robot. This is very simple in the normal case. During execution, the sensed data is matched with the left hand side of the rules. A rule for which all terms $s_i = v_i$ in the left hand side match the sensed data is said to *fire*. Three cases can occur:

1. Exactly one rule fires. This case is easy. The right hand side $y = a$ of the rule is used to control the robot.

2. More than one rule fires. The situation can be viewed as command arbitration or command fusion, and different strategies are feasible:

   (a) One way to do command arbitration is to select the rule with highest *confidence* or *strength* (see Section 2). A better alternative, judging from our experiments, is to use a majority vote among all rules that fire.

(b) Simple command fusion is possible for scalar responses by computing the average response for all rules that fire. A strength-weighted average is another possibility. For discrete responses, the average can be rounded to the nearest discrete response value.

3. No rule fires. The task of finding a rule for sensor data that lies outside all defined rules can be viewed as a classification problem: Which rule does the sample belong to? We have successfully designed and implemented a method called *k-nearest rules*, based on the classification technique k-nearest neighbors (kNN). By defining a distance between the sensor data sample and the left hand sides of the rules, the nearest rule can be determined. This rule is then considered being in a fire state, and the control signal is generated according to case 1 above. Applying the algorithm with $k > 1$ is also possible and will results in a case 2 situation. In the experiments presented in this paper, $k = 1$ has been used. The method is described in more detail in Section 5.

Experimental results for the various cases and approaches described above are presented in the examples in the next section.

4 Experiments

The experiments will use the "Programming by Demonstration" paradigm common in robotics. An overview and survey can be found in [6]. In short, our experiments are divided into three steps:

1. The robot is controlled either by a human teacher who remote control the robot, or by a manually coded controller to perform a certain task. The sensor data $S(t)$ is recorded along with the commanded response signal $R(t)$.

2. The recorded data is used in a modeling, where a control law $B : S(t) \rightarrow R(t)$ is created. In our case B is represented by a rule base with association rules as described in Section 3.1.

3. The controller B is implemented in the robot, which hopefully manages to perform the demonstrated task autonomously.

4.1 Experiment 1

The first experiment will use a manually programmed controller to teach the robot a left-wall-following behavior. The pseudo code for the controller is:

```
if      IR1 > 300
        vl = 5
        vr = −5
elseif  IR0 < 900
        vl = 2
        vr = 5
else
        vl = 5
        vr = 5
end
```
(5)

where v_l and v_r are the speed of the left and right wheel respectively. The controller is run with a cycle time of 0.1 seconds for 50 seconds. This results in 500 samples of training data, each sample consisting of 8 sensor read-outs $IR_0, IR_1, .., IR_7$ and one action y. These values are discretized as described in (2) and (3). The data is then used to automatically generate association rules. The rules with highest *strength* are listed in Table 1 together with *coverage*, *support*, *strength, lift* and *leverage* (refer to Section 2 for definitions).

A controller can be constructed from a subset of the generated rules. Using only the 4 rules with *strength*=1 gives a real-robot behavior that is almost indistinguishable from the hand coded original. This is a very satisfying result but makes it hard to evaluate the algorithm details described in Section 3.2. A quantitative and more objective way of evaluating different strategies and settings has therefore been adopted. Table 2 shows performance for a number of different

Table 1: Generated rule base for left wall follower. The 5 first rules are sufficient to emulate the wanted behavior.

Rule	**No.**	**Coverage**	**Support**	**Strength**	**Lift**	**Lev.**
$ir_1 = 2 \Rightarrow y = -3$	1	9	9	1.00	5.68	0.01
$ir_0 = 0 \wedge ir_1 = 0 \Rightarrow y = 1$	2	35	35	1.00	2.46	0.04
$ir_0 = 1 \wedge ir_1 = 0 \Rightarrow y = 1$	3	167	167	1.00	2.46	0.20
$ir_0 = 2 \wedge ir_1 = 0 \Rightarrow y = 0$	4	209	209	1.00	2.39	0.24
$ir_1 = 1 \Rightarrow y = -3$	5	80	79	0.99	5.61	0.13
$ir_0 = 0 \wedge ir_2 = 0 \Rightarrow y = 1$	6	36	35	0.97	2.39	0.04
$ir_0 = 1 \wedge ir_3 = 0 \wedge ir_7 = 0 \Rightarrow y = 1$	7	147	139	0.95	2.33	0.16
$ir_0 = 1 \wedge ir_2 = 0 \wedge ir_7 = 0 \Rightarrow y = 1$	8	140	132	0.94	2.32	0.15
$ir_2 = 2 \Rightarrow y = -3$	9	13	12	0.92	5.24	0.02
$ir_0 = 2 \Rightarrow y = 0$	10	228	209	0.92	2.19	0.23
$ir_1 = 0 \wedge ir_2 = 1 \Rightarrow y = 1$	11	23	21	0.91	2.25	0.02
$ir_0 = 1 \wedge ir_2 = 0 \Rightarrow y = 1$	12	162	145	0.90	2.20	0.16
$ir_3 = 2 \Rightarrow y = -3$	13	9	8	0.89	5.05	0.01
$ir_0 = 1 \wedge ir_3 = 0 \Rightarrow y = 1$	14	177	156	0.88	2.17	0.17
$ir_4 = 1 \Rightarrow y = -3$	15	14	12	0.86	4.87	0.02
$ir_0 = 0 \Rightarrow y = 1$	16	42	35	0.83	2.05	0.04
$ir_0 = 1 \wedge ir_7 = 0 \Rightarrow y = 1$	17	176	145	0.82	2.03	0.15
$ir_1 = 0 \wedge ir_7 = 2 \Rightarrow y = 1$	18	11	9	0.82	2.02	0.01
$ir_0 = 1 \wedge ir_6 = 0 \Rightarrow y = 1$	19	198	158	0.80	1.97	0.16
$ir_2 = 0 \wedge ir_5 = 1 \Rightarrow y = 0$	20	29	22	0.76	1.81	0.02
$ir_0 = 1 \Rightarrow y = 1$	21	230	168	0.73	1.80	0.15
$ir_3 = 1 \Rightarrow y = -3$	22	55	40	0.73	4.13	0.06
$ir_2 = 1 \Rightarrow y = -3$	23	69	45	0.65	3.71	0.07
$ir_5 = 1 \Rightarrow y = 0$	24	41	23	0.56	1.34	0.01
$ir_6 = 1 \Rightarrow y = -3$	25	43	24	0.56	3.17	0.03
$ir_7 = 1 \Rightarrow y = -3$	26	55	30	0.55	3.10	0.04
$ir_1 = 0 \Rightarrow y = 0$	27	411	209	0.51	1.22	0.07
$ir_2 = 0 \Rightarrow y = 0$	28	418	207	0.50	1.18	0.06
$ir_1 = 0 \Rightarrow y = 1$	29	411	202	0.49	1.21	0.07
$ir_3 = 0 \Rightarrow y = 0$	30	436	205	0.47	1.12	0.05
$ir_7 = 0 \Rightarrow y = 0$	31	427	196	0.46	1.10	0.04
$ir_7 = 2 \Rightarrow y = -3$	32	18	7	0.39	2.21	0.01
$ir_0 = 1 \Rightarrow y = -3$	33	230	62	0.27	1.53	0.04

Table 2: Performance for left-wall follower. Strength weighted average actions are used when more than one rule fires.

Strength	**#rules**	$\mathbf{e}_{tr}$**%**	$\mathbf{e}_{te}$**%**	**0rule%**	**1rule%**	**2rules%**	**3rules%**	**>3rules%**
1.00	4	0.2	0.1	16.1	83.9	0.0	0.0	0.0
0.98	5	0.2	0.1	0.0	100.0	0.0	0.0	0.0
0.90	11	6.6	7.6	0.0	11.2	60.9	27.9	0.0
0.80	18	15.2	16.7	0.0	0.8	50.5	17.6	31.1
0.70	22	20.8	21.5	0.0	0.0	38.5	16.2	45.3

Table 3: Performance for left-wall follower. Majority voting is used when more than one rule fires.

Strength	**#rules**	$\mathbf{e}_{tr}$**%**	$\mathbf{e}_{te}$**%**	**0rule%**	**1rule%**	**2rules%**	**3rules%**	**>3rules%**
1.00	4	0.2	0.1	16.1	83.9	0.0	0.0	0.0
0.98	5	0.2	0.1	0.0	100.0	0.0	0.0	0.0
0.90	11	1.4	1.1	0.0	11.2	60.9	27.9	0.0
0.80	18	4.0	4.3	0.0	0.8	50.5	17.6	31.1
0.70	22	6.4	6.9	0.0	0.0	38.5	16.2	45.3

controllers with different numbers of rules. The number of rules is set by giving a lower limit for the *strength* value. Each controller is evaluated on one row in the table. The rules are applied to two data sets, the 500 samples big training data set which was used to generate the rules, and a test data set which consists of 1000 samples computed at a different occasion. The e_{tr} and e_{te} are the fraction of samples that gives incorrect action when compared to the manually programmed controller 5.

The situations when more than one rule fires are handled by computing the *strength* weighted average action for the rules that fire. The situation when no rule fires is solved by the 1-nearest rule method described in Section 3.2 and in more detail in Section 5.

By demanding a *strength* value equal to 1.0, 4 rules are selected. The column labeled $0rule\%$ is the fraction of samples for which no matching rule can be found in the controller's data base. These situations are handled by the 1-nearest rule method. The 4 rule controller leaves 16.1% of the samples not matched by any rule. The 1-nearest rule handles this very well and results in 0.1% incorrect actions on the test data set.

The column labeled $1rule\%$ is the fraction of samples which are covered by exactly one rule. The rightmost 3 columns are the fractions of samples which are covered by 2, 3 and more than 3 rules respectively. These situations are handled by computing the *strength* weighted average action for the rules that cover the sample in question. This average is then used as control signal. For *strength*=0.9, 11 rules are selected and the number of samples not covered by any rule is 0.0%. 60.9% of the samples are covered by 2 rules. The average of the rules that fire is used as action, and the overall error increases to 7.6%. Adding more rules to the controller's data base results in more ambiguity, and the error increases as average actions for more and more rules are computed.

The error for the training set is slightly lower than for the test set for all controllers. This is the normal behavior with many machine learning techniques such as neural networks. However, the difference is very small and there appears to be no risk for overfitting the rule base, i.e. generating rules that are only valid for the specific data in the training data set.

An alternative strategy for resolving the situations when many rules fire simultaneously is evaluated in Table 3. A majority voting among the rules that fire is used to generate the controller's action. The error for the 11-rules controller is reduced from 7.6% to 1.1%. The errors for the controllers with more rules are also significantly reduced. The method with majority voting can therefore be said to be superior to the *strength* weighted average action method used in Table 2.

The function of the generated rules are often hard to comprehend, even when they contain few terms. A graphical presentation of the rules is therefore informative. Figure 2 shows rules 1-5 from Table 1. These 5 rules reproduce the behavior of the hand coded controller in (5) and also covers the input space completely. Referring to the coding of sensors (2) and actions (3), the function of the 5 rules can be understood. The rules $ir_1 = 1 \Rightarrow y = -3$ and $ir_1 = 2 \Rightarrow y = -3$, are responsible for sharp clockwise turns away from the wall to the left of the robot. Note that these rules cover the entire horizontal ir_0 axis, since the rules themselves do not contain the term ir_0. The rule $ir_0 = 0 \wedge ir_1 = 0 \Rightarrow y = 1$, is responsible for soft anti clockwise turns when the robot loses contact with the wall. The rule $ir_0 = 1 \wedge ir_1 = 0 \Rightarrow y = 1$, has the same function but for shorter distances measured by ir_0. Finally, the rule $ir_0 = 2 \wedge ir_1 = 0 \Rightarrow y = 0$, is responsible for driving straight ahead, parallel to the wall, when the robot is close to the wall but not turned too much towards it.

4.2 Experiment 2

This experiment deals with the Road sign problem [12], in which the robot has to act on a road sign it passed earlier. It is impossible to achieve this in a purely reactive manner since the robot has to choose between a left and a right turn depending on past stimuli. The situation is illustrated in Figure 3.

Our approach is to let the robot act on preprocessed sensor data with a perceptual decay [24]. The perception of a road sign remains even after the stimuli has disappeared and slowly fades out with time. In this way the behavior can still be purely reactive since the memory is hidden in the robot's perception. This is indeed a simplification of the original road sign problem, but it serves our purpose well. The purpose of the experiment is to see how a complex behavior can be modeled by the rule base of automatically generated association rules.

The behavior is manually coded as a switching between two controllers, a left-wall follower and a right-wall follower. The switching occurs when the robot encounters a road signs, describing

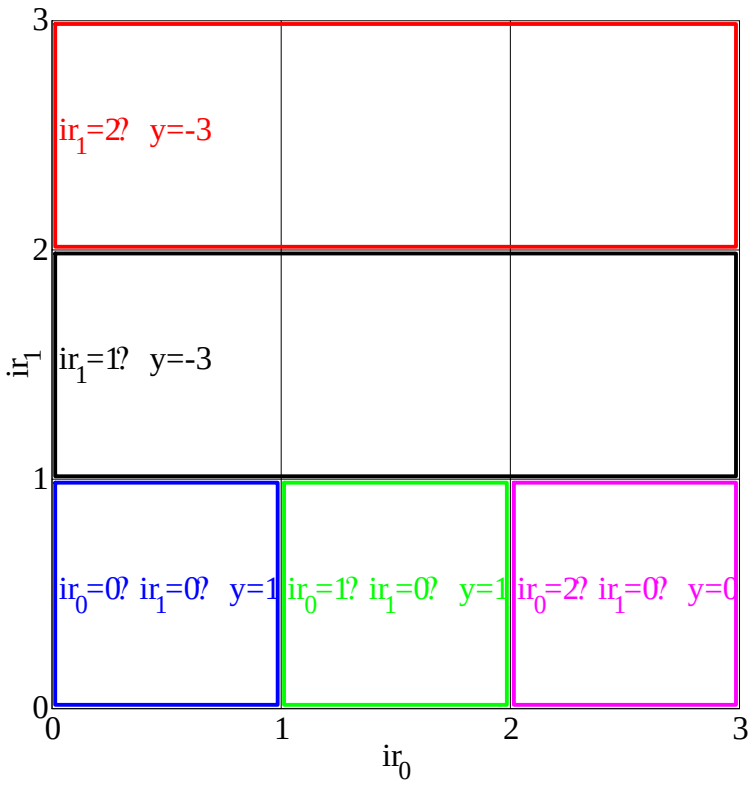

Figure 2: Graphical representation of rule 1-5 from Table 1. These 5 rules are automatically generated and reproduce the behavior of the hand coded controller in program listing (5).

the recommended way to go in the upcoming junction. The road signs are constructed of small bulbs attached to the walls of the robot's maze. The bulbs on the wall are sensed by the ambient light sensors on the Khepera robot. The sensor for left and right bulb detection are denoted AL_l and AL_r respectively. To make it possible for the robot to act on a road sign that appears and disappears before a junction, a virtual road sign sensor RS is defined as:

$$RS = \begin{cases} 2 \text{ if } & decay(AL_l) > decay(AL_r) \\ 0 & otherwise \end{cases} . \tag{6}$$

The *decay* function computes a perceptual decay of the sensed road sign signal, and serves to make the robot gradually forget about road signs as time passes after the road sign has disappeared out of sight for the robot. The RS sensor is a binary signal with the value 2 if the last seen road sign was a left sign, and 0 otherwise. The perceptual decay is a slight side step from a pure reactive design, but is a neat way of stretching the borders of the reactive paradigm when the robot's action has to depend on "old" sensor data. In our example, the RS signal will be added to the 8 infrared sensors $ir_0, ir_1, .., ir_7$ as an additional input, and will serve as a switch between the two wall followers in the learning mode. The manually programmed controller performs a left/right wall following task and is described by the following pseudo code:

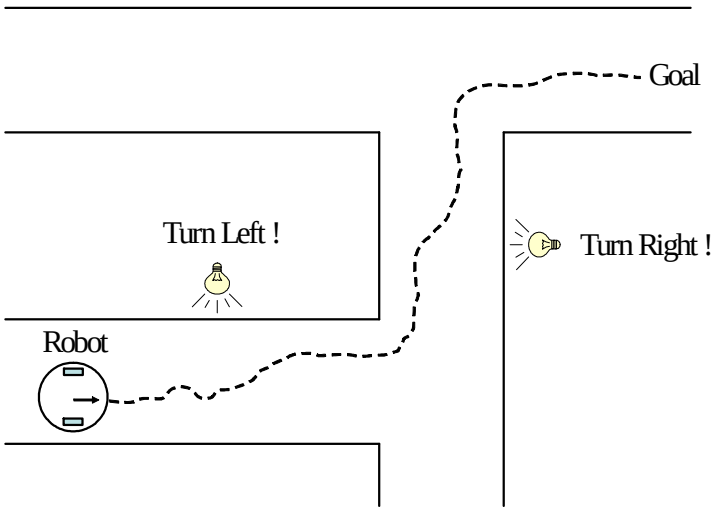

Figure 3: The road sign problem, adapted from [12], in which the robot has to decide on a left or right turn in each junction, depending on the past stimuli from the road signs. Our approach is to add a perceptual decay to the road sign perception. The robot switches between a left and right wall following behavior.

```
if    RS = 2
       if      IR1 > 300
               vl = 5
               vr = −5
       elseif  IR0 < 900
               vl = 2
               vr = 5
       else
               vl = 5
               vr = 5
       end
else                                                    (7)
       if      IR4 > 300
               vl = −5
               vr = 5
       elseif  IR3 < 900
               vl = 5
               vr = 2
       else
               vl = 5
               vr = 5
       end
end
```

where v_i and v_r are the speeds of the left and right wheel respectively. The upper block in the code is the same as for Example 1, and performs a left-wall following behavior, while the lower block performs a right-wall following behavior. In step 2 of the "Programming by Demonstration" method, RS is made available as an extra input in the search for association rules, and should then (automatically) be added as a high level condition that groups the generated rules into two categories: left-wall following and right-wall following. Rules common to both behaviors may of course be unaffected by the value of the RS input.

The controller is run with a cycle time of 0.1 seconds for 100 seconds. This results in 1000 samples of training data, each sample consisting of 8 discretized sensor read-outs ir_{0-7}, RS and one discretized action y. The discretization is described in (2) and (3). The data is then used

to automatically generate association rules. The rules with highest *strength* are listed in Table 4 together with *strength* and *confidence*.

Table 5 shows performance in the same way as presented for Example 1 above. Different numbers of rules are selected depending on a lower set limit for the *strength* value. Performance for the training data set and, for a separately recorded 5000 samples long test data set is presented. For *strength*= 1, 9.6% of the test data has sensor data such that no rule fires. This is, as before, resolved by the 1-nearest rule method. In 23.9% of the cases, two or more rules fire at the same time. This is resolved by majority voting among the rules that fire. It is clear from the table that the best controller is achieved from the 43 rules with *strength*$\geqslant 0.95$. These rules give minimum error on both training and test data. Furthermore, the number of cases where no rule fires is reduced to zero when these 43 rules are used.

A comparison between the training set error e_{tr} and test set error e_{te} exhibits a difference that would normally be diagnosed as overfitting. This concept is largely ignored in the association rule community [7], while it is very common in other areas of machine learning. In our opinion, overfitting is a problem that has to be taken into account also when generating association rules. Acting on rules with very low *strength* or *support* corresponds to adding more nodes to an artificial neural network, or adding higher degree terms to a polynomial model. Simple techniques, such as computing performance for both training data and previously unseen test data should therefore be a standard procedure when applying association rules to most domains, in particular with noisy data such as robot applications.

Table 4 shows that not all rules responsible for turning (i.e. rules where $y \neq 0$) contain the RS variable as condition on the left hand side of the rule. However, this is not necessarily incorrect, since turning may occur not only when performing a turn in a junction but also for wall avoidance, which could be handled uniformly, regardless of the road sign condition. When installed as a controller on a real Khepera robot, the robot successfully manages to switch between left and right wall following depending on road marks placed along the route in the maze. Considerably more rules are required to achieve a successful behavior than was the case in the simple task in Experiment 1. This is caused by the much harder situation in the rule generation phase. All learned rules for junction turning has to include the RS variable. Even if accurate rules can be generated, the *coverage* for these rules are relatively low.

Figure 4 shows rules 22 and 41 from Table 4. These rules are examples of how the system learns how to act differently on the same sensor, depending on the value of the road sign variable RS. The rule $ir_0 = 2 \wedge RS = 0 \Rightarrow y = -1$, is responsible for soft clockwise turns away from the left wall if it gets too close when in right-wall following mode. The rule $ir_0 = 0 \wedge RS = 2 \Rightarrow y = 1$, is responsible for soft anti clockwise turns toward the left wall if it gets too far away from it when in left-wall following mode.

4.3 Experiment 3

As previously mentioned, the ease by which a 5-rule controller can be automatically generated in Example 1 is partly due to the favorable partitioning of the input space. This is illustrated in Figure 5. The manually defined rule borders (2) match the break points in the program (5) exactly. This is of course an advantage when generating association rules to emulate the program, and some rules get both high *support* and *strength* values. In this experiment we will therefore approach the problem without any prior knowledge of the sensor characteristics or about the task at hand. Each sensor readout IR_i is split into 4 ranges $0, 1, 2, 3$ and represented by a discrete variable ir_i according to

$$ir_i = \begin{cases} 0 \text{ if } & 0 \leq IR_i < 250 & \text{(very long distance)} \\ 1 \text{ if } & 250 \leq IR_i < 500 & \text{(long distance)} \\ 2 \text{ if } & 500 \leq IR_i \leq 750 & \text{(medium distance)} \\ 3 \text{ if } & 750 \leq IR_i \leq 1023 & \text{(short distance)} \end{cases} . \tag{8}$$

As before, the data is used to generate association rules. The resulting rule base is shown in Table 6, and reveals that the situation is much harder with the blind partitioning. The *strength* and also *coverage* of the found rules are much lower than in Table 3 for Example 1. The reason for this is that the data for a specific condition, such as "too close to left wall", more often gets spread over boundaries in the partitioning of the input space. The performance for the generated

Table 4: Generated rule base for road sign controller. The binary RS variable controls left and right wall following.

Rule	**No.**	**Coverage**	**Support**	**Strength**	**Lift**	**Lev.**
$ir_1 = 2 \wedge ir_2 = 2 \Rightarrow y = -3$	1	4	4	1.00	13.70	0.00
$ir_0 = 1 \wedge ir_1 = 2 \Rightarrow y = -3$	2	4	4	1.00	13.70	0.00
$ir_1 = 2 \wedge RS = 2 \Rightarrow y = -3$	3	6	6	1.00	13.70	0.01
$ir_0 = 1 \wedge ir_2 = 1 \wedge ir_6 = 1 \Rightarrow y = -3$	4	3	3	1.00	13.70	0.00
$ir_1 = 1 \wedge RS = 2 \Rightarrow y = -3$	5	67	67	1.00	13.70	0.06
$ir_2 = 2 \wedge ir_3 = 0 \Rightarrow y = -3$	6	2	2	1.00	13.70	0.00
$ir_2 = 1 \wedge ir_6 = 2 \wedge ir_7 = 1 \Rightarrow y = 3$	7	2	2	1.00	10.53	0.00
$ir_4 = 2 \wedge ir_6 = 1 \Rightarrow y = 3$	8	7	7	1.00	10.53	0.01
$ir_4 = 2 \wedge ir_7 = 1 \Rightarrow y = 3$	9	2	2	1.00	10.53	0.00
$ir_2 = 1 \wedge ir_4 = 1 \Rightarrow y = 3$	10	26	26	1.00	10.53	0.02
$ir_1 = 1 \wedge ir_4 = 2 \Rightarrow y = 3$	11	4	4	1.00	10.53	0.00
$ir_0 = 1 \wedge ir_4 = 2 \Rightarrow y = 3$	12	2	2	1.00	10.53	0.00
$ir_4 = 2 \wedge RS = 0 \Rightarrow y = 3$	13	25	25	1.00	10.53	0.02
$ir_3 = 2 \wedge RS = 0 \Rightarrow y = 3$	14	32	32	1.00	10.53	0.03
$ir_0 = 2 \wedge ir_1 = 0 \wedge RS = 2 \Rightarrow y = 0$	15	94	94	1.00	4.13	0.07
$ir_4 = 0 \wedge ir_5 = 2 \wedge RS = 0 \Rightarrow y = 0$	16	148	148	1.00	4.13	0.11
$ir_1 = 2 \wedge RS = 0 \Rightarrow y = -1$	17	8	8	1.00	3.89	0.01
$ir_1 = 2 \wedge ir_2 = 0 \Rightarrow y = -1$	18	6	6	1.00	3.89	0.00
$ir_0 = 1 \wedge ir_5 = 1 \wedge ir_6 = 1 \Rightarrow y = -1$	19	3	3	1.00	3.89	0.00
$ir_0 = 1 \wedge ir_6 = 1 \wedge RS = 0 \Rightarrow y = -1$	20	3	3	1.00	3.89	0.00
$ir_0 = 1 \wedge ir_7 = 1 \wedge RS = 0 \Rightarrow y = -1$	21	7	7	1.00	3.89	0.01
$ir_0 = 2 \wedge RS = 0 \Rightarrow y = -1$	22	13	13	1.00	3.89	0.01
$ir_3 = 0 \wedge ir_5 = 1 \wedge ir_6 = 0 \wedge RS = 0 \Rightarrow y = -1$	23	116	116	1.00	3.89	0.09
$ir_4 = 0 \wedge ir_5 = 1 \wedge RS = 0 \Rightarrow y = -1$	24	142	142	1.00	3.89	0.11
$ir_4 = 1 \wedge RS = 2 \Rightarrow y = 1$	25	13	13	1.00	3.00	0.01
$ir_1 = 0 \wedge ir_2 = 2 \wedge RS = 2 \Rightarrow y = 1$	26	101	101	1.00	3.00	0.07
$ir_4 = 2 \wedge RS = 2 \Rightarrow y = 1$	27	133	133	1.00	3.00	0.09
$ir_0 = 1 \wedge ir_1 = 0 \wedge RS = 2 \Rightarrow y = 1$	28	128	128	1.00	3.00	0.09
$ir_1 = 0 \wedge ir_5 = 1 \wedge RS = 2 \Rightarrow y = 1$	29	43	43	1.00	3.00	0.03
$ir_5 = 2 \wedge RS = 2 \Rightarrow y = 1$	30	107	107	1.00	3.00	0.07
$ir_0 = 0 \wedge ir_1 = 0 \wedge RS = 2 \Rightarrow y = 1$	31	205	205	1.00	3.00	0.14
$ir_4 = 1 \wedge RS = 0 \Rightarrow y = 3$	32	71	70	0.99	10.38	0.06
$ir_2 = 2 \wedge ir_5 = 2 \Rightarrow y = 1$	33	60	59	0.98	2.95	0.04
$ir_3 = 2 \wedge RS = 2 \Rightarrow y = 1$	34	133	130	0.98	2.94	0.09
$ir_5 = 0 \wedge RS = 0 \Rightarrow y = -1$	35	117	114	0.97	3.79	0.08
$ir_1 = 0 \wedge ir_2 = 1 \wedge RS = 2 \Rightarrow y = 1$	36	35	34	0.97	2.92	0.02
$ir_0 = 1 \wedge ir_1 = 1 \Rightarrow y = -3$	37	33	32	0.97	13.28	0.03
$ir_2 = 2 \wedge ir_4 = 2 \Rightarrow y = 1$	38	99	96	0.97	2.91	0.06
$ir_2 = 0 \wedge ir_5 = 1 \wedge ir_6 = 0 \wedge RS = 0 \Rightarrow y = -1$	39	124	119	0.96	3.73	0.09
$ir_0 = 2 \wedge ir_1 = 0 \Rightarrow y = 0$	40	98	94	0.96	3.96	0.07
$ir_0 = 0 \wedge RS = 2 \Rightarrow y = 1$	41	214	205	0.96	2.88	0.13
$ir_0 = 1 \wedge ir_1 = 0 \wedge ir_5 = 0 \Rightarrow y = 1$	42	133	127	0.95	2.87	0.08
$ir_4 = 0 \wedge ir_5 = 1 \Rightarrow y = -1$	43	149	142	0.95	3.71	0.10

Table 5: Performance for road sign controller. Majority voting is used when more than one rule fires. The error rate is much higher than for the simple wall following task. The difference between the training error e_{tr} and test error e_{te} is an indication of overfitting.

Strength	**#rules**	$\mathbf{e_{tr}}$**%**	$\mathbf{e_{te}}$**%**	**0rule%**	**1rule%**	**2rules%**	**3rules%**	**>3rules%**
1.00	31	1.0	5.0	9.6	66.5	18.9	3.2	1.8
0.98	33	0.9	4.9	6.5	66.8	20.7	4.0	1.9
0.95	43	0.5	2.4	0.0	26.8	46.9	10.2	16.1
0.90	56	3.1	4.7	0.0	7.7	44.4	21.1	26.8
0.85	66	4.7	6.0	0.0	7.1	26.8	31.1	35.0
0.80	76	4.2	6.2	0.0	6.2	23.2	2.8	67.8
0.75	87	7.1	9.9	0.0	5.9	0.6	9.1	84.4
0.70	97	6.0	9.1	0.0	5.9	0.3	6.4	87.4

controllers are presented in Table 7. The error rates e_{tr} and e_{te} are also much higher than in Table 3 for Example 1. However, by using the 19 rules with *strength*>0.70, the robot reasonably well reproduces the left-wall following behavior, even if the error rate is as high as 27.1%.

Table 6: Generated rule base for left-wall follower. The input space is unformly partitioned into 4 parts.

Rule	**No.**	**Coverage**	**Support**	**Strength**	**Lift**	**Lev.**
$ir_7 = 3 \Rightarrow y = -3$	1	3	3	1.00	4.42	0.00
$ir_4 = 3 \Rightarrow y = -3$	2	4	4	1.00	4.42	0.01
$ir_1 = 2 \Rightarrow y = -3$	3	8	8	1.00	4.42	0.01
$ir_1 = 3 \Rightarrow y = -3$	4	16	16	1.00	4.42	0.02
$ir_1 = 1 \Rightarrow y = -3$	5	22	22	1.00	4.42	0.03
$ir_5 = 3 \Rightarrow y = 1$	6	11	11	1.00	1.98	0.01
$ir_0 = 0 \wedge ir_1 = 0 \wedge ir_2 = 0 \Rightarrow y = 1$	7	141	131	0.93	1.84	0.12
$ir_2 = 3 \Rightarrow y = -3$	8	20	18	0.90	3.98	0.03
$ir_0 = 3 \wedge ir_1 = 0 \Rightarrow y = 0$	9	92	80	0.87	3.22	0.11
$ir_0 = 0 \wedge ir_2 = 0 \Rightarrow y = 1$	10	151	131	0.87	1.72	0.11
$ir_0 = 0 \wedge ir_1 = 0 \Rightarrow y = 1$	11	158	136	0.86	1.71	0.11
$ir_3 = 3 \Rightarrow y = -3$	12	13	11	0.85	3.74	0.02
$ir_2 = 1 \Rightarrow y = -3$	13	26	21	0.81	3.57	0.03
$ir_0 = 1 \wedge ir_1 = 0 \Rightarrow y = 1$	14	136	107	0.79	1.56	0.08
$ir_0 = 2 \Rightarrow y = 0$	15	70	55	0.79	2.91	0.07
$ir_0 = 3 \Rightarrow y = 0$	16	104	80	0.77	2.85	0.10
$ir_0 = 0 \Rightarrow y = 1$	17	177	136	0.77	1.52	0.09
$ir_2 = 2 \Rightarrow y = -3$	18	4	3	0.75	3.32	0.00
$ir_0 = 1 \Rightarrow y = 1$	19	149	107	0.72	1.42	0.06
$ir_3 = 1 \Rightarrow y = -3$	20	13	9	0.69	3.06	0.01
$ir_1 = 0 \Rightarrow y = 1$	21	454	252	0.56	1.10	0.05

Some of the generated rules are graphically shown in Figure 5 together with the raw data. It is obvious that it takes more rules to reproduce a behavior if the boundaries of the rules do not coincide with the process that generates the behavior. Suggested methods to adjust the boundaries are discussed in Section 6.

5 The k-nearest rules method

This section will describe the k-nearest rules method briefly introduced in Section 3.2. The method is used in cases when a certain sample does not match any of the rules in the rule base (i.e.: no rule "fires"). As mentioned before, the task of finding a rule for such a sample can be viewed as a classification problem: Which rule does the sample belong to? A distance measure between the sample and the left hand sides of the rules will be defined. The nearest rule is considered being

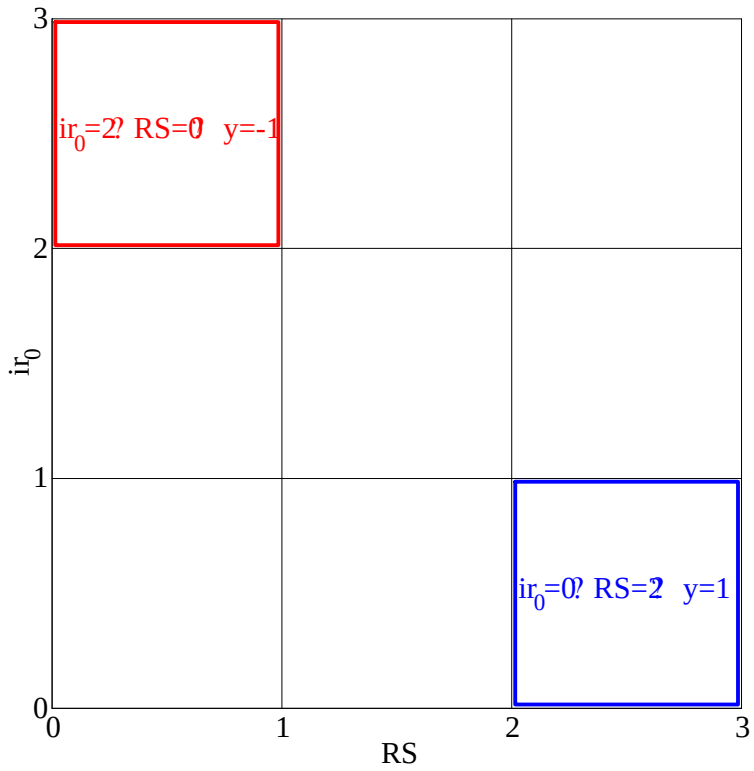

Figure 4: Graphical representation of rules 22 and 41 from Table 4. The rules demonstrate how the system learns how to act differently on sensor data depending on the road sign variable RS.

in a fire state, and the control signal is generated from the right hand side of that rule. Applying the k-nearest rules technique with k>1 is also possible and will results in a situation similar to case 2 described in Section 3.2. Multiple actions will then have to be fused into one, using either arbitration or command fusion. In the experiments presented in this paper, k=1 is used. The corresponding method is denoted *the 1-nearest rule method*.

The distance $D(X, R)$ between a sample $X : (X_0, ..., X_n)$ and an association rule $R : s_i = v_i \wedge ... \wedge s_j = v_j \Rightarrow y = a$ is defined as

$$D(X, R) = \sum |x_k - v_k| \tag{9}$$

where the sum goes over all terms in R, and each x_k is the discretized version of the sampled variables in X, e.g. as defined for the infrared sensors in (2). I.e., D is the sum of absolute

Table 7: Performance for left-wall follower. Majority voting is used when more than one rule fires. The input space is unformly partitioned into 4 parts.

Strength	**#rules**	$\mathbf{e}_{tr}$**%**	$\mathbf{e}_{te}$**%**	**0rule%**	**1rule%**	**2rules%**	**3rules%**	**>3rules%**
1.00	6	74.4	80.7	74.1	21.3	4.4	0.2	0.0
0.95	6	74.4	80.7	74.1	21.3	4.4	0.2	0.0
0.90	8	65.2	75.0	68.0	22.3	8.6	1.1	0.0
0.85	11	39.2	35.1	17.7	65.2	10.3	6.8	0.0
0.80	13	39.4	36.2	17.0	59.9	12.2	10.1	0.8
0.75	18	31.4	27.5	0.0	17.3	58.8	12.1	11.8
0.70	19	31.2	27.1	0.0	10.2	62.7	14.4	12.7
0.65	20	31.2	27.7	0.0	10.2	61.0	12.0	16.8
0.60	20	31.2	27.7	0.0	10.2	61.0	12.0	16.8
0.55	21	31.0	27.6	0.0	0.0	16.5	62.0	21.5

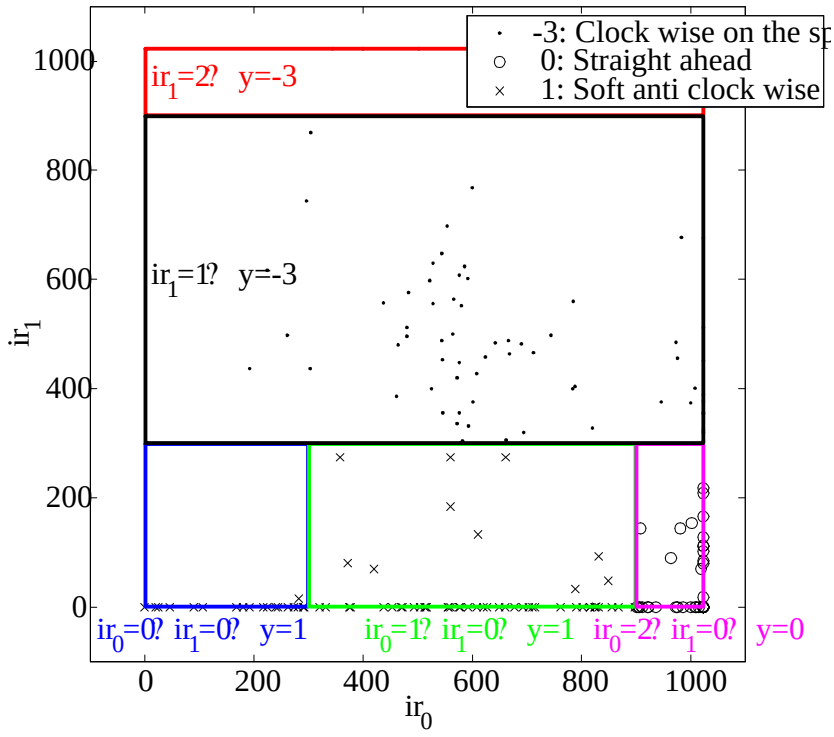

Figure 5: Graphical representation of rule 1-5 from Table 1, with raw data plotted. The rule borders coincide with the program that generated the data. This makes it relatively easy to find rules that perform the wall following behavior.

differences for all terms in the rule. The idea is illustrated in Figure 7. A sample located anywhere in the top row of the diagram will have the rule $ir_1 = 1 \Rightarrow y = -3$ as nearest neighbor (with $D = 2$), and will therefore be assigned the action $y = -3$ by the 1-nearest rule algorithm. The advantage with using discretized variables instead of raw data in the distance measure is that the effects of non-linearities and scale differences between different kinds of sensor signals are eliminated. A few other types of distance measures have been tested, but the one in (9) gives best results, at least for the tested examples.

The distance D will only take integer values ≥ 0, and many rules in the controller's rules base may have the same distance to a certain sample. In Example 3 with 6 rules, 74.1% of the samples have no rule matching them. With the 1-nearest rule method and distance measure (9), 95% of the samples have a unique nearest rule. However, this might be application dependent and techniques to deal with ambiguous nearest rules are among the suggestions for further research mentioned in Section 6. The method used in the examples in this report, handle ambiguous nearest rules by picking the one with highest *strength* (also this may very well be ambiguous, especially for *strength*=1).

6 Suggested Continued Research

From the examples presented above, we can conclude that the error in the controllers drops significantly when enough number of rules are included to cover the entire active input space, and thus reduce the $0rule\%$ cases to 0. This is an indication that the 1-nearest rule technique is inferior to adding more, possibly less accurate, rules to cover the input space. However, the 1-nearest rule technique is useful in situations when enough rules can simply not be generated from the sampled data, and some kind of method to generate an action has to be applied. As mentioned in the previous section, ways to deal with ambiguous nearest rules have to be further investigated, as well as other distance measures. Totally different ways to generate an action when no rule covers the sensor data is also a suggested topic for future research.

The successful result with a 5-rule controller being able to reproduce the wall following behavior is partly caused by the favorable choice of discretization of the input space (2). The attempt with

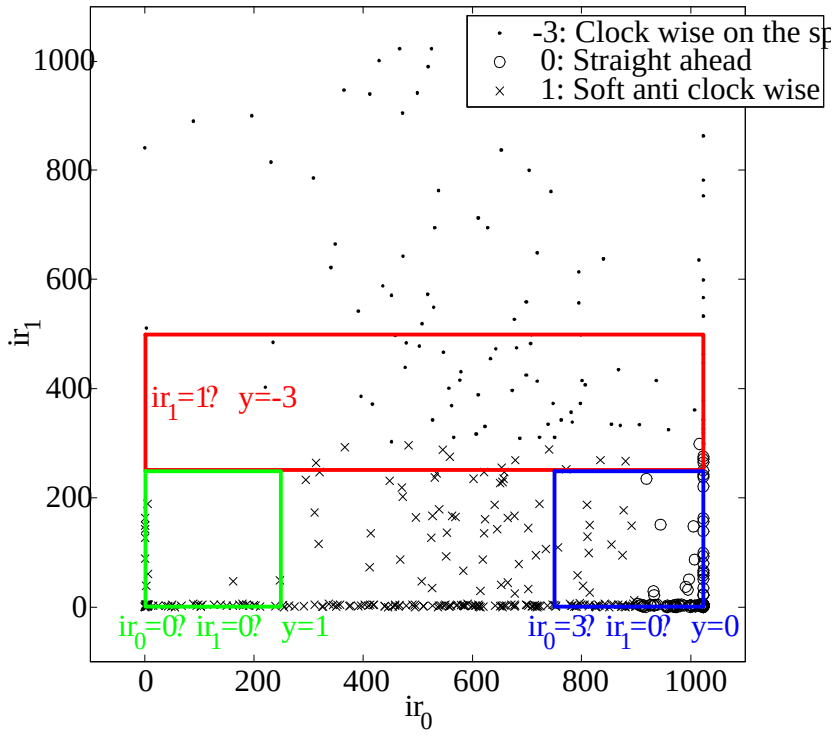

Figure 6: Graphical representation of the rules 5, 9 and 11 from Table 6, with raw data plotted. The rule borders do not coincide with the program that generated the data. This makes it necessary to use more rules to achieve the wall following behavior.

uniform discretization in Example 3 clearly shows this fact, but still results in controllers that successfully reproduce the wall following task, even if the required number of rules gets higher. An interesting continuation of this work is to develop techniques for tuning the generated rules such that the borders in the partitioning become optimized (either on the rule base, rule or variable level).

The proposed technique with association rules to represent *stimuli* $\rightarrow$ *respons* mappings for reactive robots resembles fuzzy logic controllers in many respects. While not including any fuzzy elements, the association rules have the same form (1) as fuzzy rules commonly used in reactive control [20, 9]. One can easily imagine extending the association rules to fuzzy counterparts. This has already been proposed for association rules in general [5], and the same techniques could be adapted to suit robotics applications. However, the algorithms for generation of ordinary crisp association rules are indeed powerful, and manage to mine data with high dimensionality and huge amounts of data records. The crisp association rules should therefore be good starting points for generation of fuzzy rules. A proposed continuation of the presented work is to look at ways of fuzzifying the border lines in the discrete association rules.

Finally, association rules have possible applications in many other areas in the field of intelligent robotics. In this paper we have focused on finding stimuli-response mappings of the form $S \rightarrow R$. This is a restricted form of association rules and does not express the full applicability of the technique. The more general form $X \Rightarrow Y$, where both X and Y are conjunctions of items can be used for a number of interesting learning tasks such as

- How the robot interacts with the world
- Common patterns in the robot's actions
- Common patterns in the robot's perception of the world
- How the robot perceives the world
- How the world evolves in time
- How innate or hard coded behaviors work

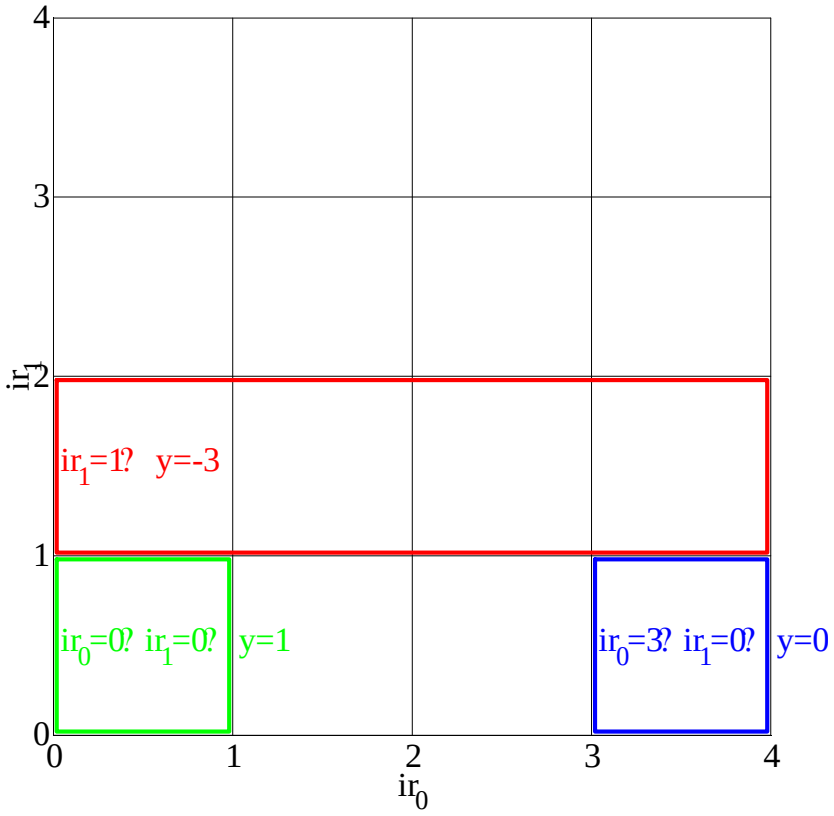

Figure 7: Illustration of the distance measure used for the k nearest rules method. Assuming a rule base with the 3 shown rules, a sample located anywhere in the top row of the diagram has the rule $ir_1 = 1 \Rightarrow y = -3$ as nearest neighbor and will therefore be assigned the action $y = -3$ by the 1-nearest rule algorithm.

7 Summary

We have demonstrated how association rules can provide a powerful alternative for intelligent robotic controllers. The presented technique automatically generates compact sets of rules that successfully control physical robots in a simple wall following task as well as in a more complex road sign application. The situations when no rule fires are resolved by the 1-nearest rule technique. The problem with overlapping rules is resolved with majority voting among all rules that fire. This technique is shown to be superior to using the strength weighted average action. A number of areas in intelligent robotics are suggested as suitable for using association rules. Much remains to be done to exploit these possibilities.

8 Acknowledgments

This work was financed by the Kempe Foundations as part of their generous support of the IFOR project. We acknowledge their support gratefully.

References

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., 26–28 1993.

[2] C. Becquet, S. Blachon, B. Jeudy, J.-F. Boulicaut, and O. Gandrillon. Strong-association-rule mining for large-scale gene-expression data analysis: a case study on human sage data. *Genome Biology*, 3(12), 2002.

[3] R. Braunstingl and J. Uribe . A wall following robot with a fuzzy logic controller optimized by a genetic algorithm. In *Proceeedings Fourth IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'95)*, volume 5.

[4] T. Brijs, G. Swinnen, K. Vanhoof, and G. Wets. Using association rules for product assortment decisions: A case study. In *Knowledge Discovery and Data Mining*, pages 254–260, 1999.

[5] K. C. Chan and W.-H. Au. Mining fuzzy association rules. In *Proc. of the Sixth International Conference on Information and Knowledge Management (CIKM'97)*, pages 209–215, 1997.

[6] R. Dillmann, O. Rogalla, M. Ehrenmann, R. Zllner, and M. Bordegoni. Learning robot behaviour and skills based on human demonstration and advice. In *Proceedings of the 9th International Symposium of Robotics Research (ISRR'99)*, 1999.

[7] A. A. Freitas. Understanding the crucial differences between classification and discovery of association rules - a position paper. *SIGKDD Explorations*, 2(1):65–69, 2000.

[8] G I Webb and Associates. *Magnum Opus - Software for Association Rules*. Web page: www.giwebb.com, Aug 2003.

[9] J. Godjevac and N. Steele. Neuro fuzzy control for basic mobile robot behaviours. In *Fuzzy Logic Techniques for Autonomous Vehicle Navigation*, pages 97–118. Springer-Verlag, 2001.

[10] J. Hipp, U. Güntzer, and G. Nakhaeizadeh. Algorithms for association rule mining – a general survey and comparison. *SIGKDD Explorations*, 2(1):58–64, July 2000.

[11] W. Lee, S. J. Stolfo, and K. W. Mok. A data mining framework for building intrusion detection models. In *IEEE Symposium on Security and Privacy*, pages 120–132, 1999.

[12] F. Linker and H. Jacobsson. Mobile robot learning of delayed response tasks through event extraction: A solution to the road sign problem and beyond. *International Journal of Computational Intelligence and Applications*, 4(1):413–426, 2001.

[13] P. Martin and U. Nehmzow. Programming by teaching: Neural network control in the manchester mobile robot. In *Proceedings Intelligent Autonomous Vehicles*, 1995.

[14] O. Miglino, H. Lund, and S. Nolfi. Evolving mobile robots in simulated and real environments. *Artificial Life 2*, pages 417–434, 1995.

[15] F. Mondada and D. Floreano. Evolution of neural control structures: Some experiments on mobile robots. *Robotics and Autonomous Systems*, 1996.

[16] U. Nehmzow. Autonomous acquisition of sensor-motor couplings in robots. Technical Report UMCS-94-11-1, Department of Computer Science, University of Manchester, 1994.

[17] C. Ordonez and E. Omiecinski. Discovering association rules based on image content. In *Proceedings of the IEEE Advances in Digital Libraries*.

[18] P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma. Evolutionary learning of a neural robot controller. *Computational Intelligence for Modelling Control*, 2001.

[19] K. Wang, S. Zhou, J. M. S. Yeung, and Q. Yang. Mining customer value: From association rules to direct marketing. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE03)*, 2003.

[20] K. Ward, A. Zelinsky, and P. McKerrow. Learning robot behaviours by extracting fuzzy rules from demonstrated actions. *Australian Journal of Intelligent Information Processing Systems*, 2001.

[21] G. I. Webb. Opus: An efficient admissible algorithm for unordered search. *Journal of Artificial Intelligence Research*, 3:431–465, 1995.

[22] G. I. Webb, S. Butler, and D. Newlands. On detecting differences between groups. In *Accepted for publication in the Proceedings of The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*, page tbc. AAAI Press, 2003.

[23] G. I. Webb and S. Zhang. Beyond association rules: Generalized rule discovery. Submitted for publication, 2003.

[24] B. B. Werger. Cooperation without deliberation: A minimal behavior-based approach to multi-robot teams. *Artificial Intelligence*, 110(2):293–320, 1999.

[25] T. Ziemke. Remembering how to behave: Recurrent neural networks for adaptive robot behavior. In *Recurrent Neural Networks: Design and Applications*. CRC Press, Boca Raton, 1999.