### Learning Robotic Behaviors with Association Rules

THOMAS HELLSTRÖM Department of Computing Science Umeå University 901 87 Umeå SWEDEN thomash@cs.umu.se http://www.cs.umu.se/~thomash

*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.

*Key-Words:* - Learning robots, Association rules, Reactive behaviors, Intelligent robotics, Machine learning, Autonomous robotics

### 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, neural networks, genetic algorithms and reinforcement learning.

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. 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. Section 5 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 *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 liftand *leverage* are useful. For a definition of these concepts see [3].

### 3 the Learning Problem

In the experiments in this paper, control of the robot is purely reactive and is defined by a control law  $B: S(t) \to R(t)$ , 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

$$s_i = v_i \wedge s_j = v_j \dots \wedge s_k = v_k \Rightarrow y = a.$$
(1)

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

$$ir_{i} = \begin{cases} 0 \text{ if } 0 \leq IR_{i} < 300 & (\text{long distance}) \\ 1 \text{ if } 300 \leq IR_{i} < 900 & (\text{medium dist.}) \\ 2 \text{ if } 900 \leq IR_{i} \leq 1023 & (\text{short distance}) \end{cases}$$

$$(2)$$

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:

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

The rule should be interpreted as follows:

if 
$$0 \le IR_1 < 300 \land 300 \le IR_2 < 900$$
 then  $v_l = 2$  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

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

$$ir_0, ir_1, ..., ir_7, y$$
 (4)

where each  $ir_i$  is an infrared sensor readout and yis 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  $S \to R$ .

Algorithms that efficiently search large databases for association rules have been previously developed [1]. We utilize a standard software Magnum Opus, based on the OPUS algorithm [5], 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.

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. For reference, all five quality measures are presented.

### 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 = 1has been used. More information about this technique can be found in [3].

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 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  $B: S(t) \to R(t)$  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:

$$\begin{array}{ll} \text{if} & \mathrm{IR}_{1} > 300 \\ & \mathrm{v}_{l} = 5; \mathrm{v}_{r} = -5 \\ \text{elseif} & \mathrm{IR}_{0} < 900 \\ & \mathrm{v}_{l} = 2; \mathrm{v}_{r} = 5 \\ \text{else} \\ & \mathrm{v}_{l} = 5; \mathrm{v}_{r} = 5 \\ \text{end} \end{array}$$

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 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.

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 1 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  $ir_0 = 0 \land ir_1 = 0 \Rightarrow y = 1$ , 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 [4], 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 2.

Our approach is to let the robot act on preprocessed sensor data with a perceptual decay [6]. 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 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:

$$RS = \begin{cases} 2 \text{ if } decay(AL_l) > decay(AL_r) \\ 0 & otherwise \end{cases}$$
 (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 as described by the following pseudo code:

where the *left-wall follower* is the same as for Example 1 and the *right-wall follower* is coded in a mirrored fashion. 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 0.1seconds 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. Table 4 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 1nearest 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  $\geq 0.95$ . 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 [2], 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 5 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. For a more detailed analysis of the road sign experiment, see [3].

## 5 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.

## 6 Acknowledgements

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

# References

- 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] A. A. Freitas. Understanding the crucial differences between classification and discov-

ery of association rules - a position paper.  $SIGKDD \ Explorations, 2(1):65-69, 2000.$ 

- [3] T. Hellström. Association rules for learning behavioral mappings in robotics. Technical Report UMINF-03. ISSN-0348-0542, Department of Computing Science Umeå University, Umeå Sweden, 2003.
- [4] F. Linåker 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.
- [5] G. I. Webb. Opus: An efficient admissible algorithm for unordered search. *Journal of Artificial Intelligence Research*, 3:431–465, 1995.
- [6] B. B. Werger. Cooperation without deliberation: A minimal behavior-based approach to multi-robot teams. *Artificial Intelligence*, 110(2):293–320, 1999.

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	$\mathbf{Lift}$	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
$ir_0 = 0 \land ir_2 = 0 \Rightarrow y = 1$	6	36	35	0.97	2.39	0.04
$ir_0 = 1 \land ir_3 = 0 \land ir_7 = 0 \Rightarrow y = 1$	7	147	139	0.95	2.33	0.16
$ir_0 = 1 \land ir_2 = 0 \land ir_7 = 0 \Rightarrow y = 1$	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

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%	2 rules %	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

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

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



Figure 1: 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).

Table 4: 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



Figure 2: The road sign problem, adapted from [4], 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.

Rule	No.	Coverage	Support	Strength	$\mathbf{Lift}$	Lev.
$ir_1 = 2 \land ir_2 = 2 \Rightarrow y = -3$	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
$ir_1 = 2 \land RS = 2 \Rightarrow y = -3$	3	6	6	1.00	13.70	0.01
$ir_0 = 1 \land ir_2 = 1 \land ir_6 = 1 \Rightarrow y = -3$	4	3	3	1.00	13.70	0.00
$ir_1 = 1 \land RS = 2 \Rightarrow y = -3$	5	67	67	1.00	13.70	0.06
$ir_2 = 2 \land ir_3 = 0 \Rightarrow y = -3$	6	2	2	1.00	13.70	0.00
$ir_2 = 1 \land ir_6 = 2 \land ir_7 = 1 \Rightarrow y = 3$	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
$ir_2 = 1 \land ir_4 = 1 \Rightarrow y = 3$	10	26	26	1.00	10.53	0.02
$ir_1 = 1 \land ir_4 = 2 \Rightarrow y = 3$	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
$ir_4 = 2 \land RS = 0 \Rightarrow y = 3$	13	25	25	1.00	10.53	0.02
$ir_3 = 2 \land RS = 0 \Rightarrow y = 3$	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
$ir_4 = 0 \land ir_5 = 2 \land RS = 0 \Rightarrow y = 0$	16	148	148	1.00	4.13	0.11
$ir_1 = 2 \land RS = 0 \Rightarrow y = -1$	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
$ir_0 = 1 \land ir_5 = 1 \land ir_6 = 1 \Rightarrow y = -1$	19	3	3	1.00	3.89	0.00
$ir_0 = 1 \land ir_6 = 1 \land RS = 0 \Rightarrow y = -1$	20	3	3	1.00	3.89	0.00
$ir_0 = 1 \land ir_7 = 1 \land RS = 0 \Rightarrow y = -1$	21	7	7	1.00	3.89	0.01
$ir_0 = 2 \land RS = 0 \Rightarrow y = -1$	22	13	13	1.00	3.89	0.01
$ir_3 = 0 \land ir_5 = 1 \land ir_6 = 0 \land RS = 0 \Rightarrow y = -1$	23	116	116	1.00	3.89	0.09
$ir_4 = 0 \land ir_5 = 1 \land RS = 0 \Rightarrow y = -1$	24	142	142	1.00	3.89	0.11
$ir_4 = 1 \land RS = 2 \Rightarrow y = 1$	25	13	13	1.00	3.00	0.01
$ir_1 = 0 \land ir_2 = 2 \land RS = 2 \Rightarrow y = 1$	26	101	101	1.00	3.00	0.07

Table 5: Part of generated rule base for road sign controller. The binary RS variable controls left and right wall following.