Teaching a Robot to Behave like a Cockroach

THOMAS HELLSTRÖM*
*Department of Computing Science
Umeå University
Sweden
thomash@cs.umu.se

Abstract

Techniques for learning reactive robot behaviors have been an active field of research in robotics for many years. In this paper the method for representing behaviors is based on association rules. Learning the association rules is accomplished by recording 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. Reactive behaviors have obvious limitations, caused by the reactivity itself. Sequences of behaviors are hard to model, unless the switch between behaviors is synchronous with changes in the sensor data. Two ways to get around this limitation are discussed, and the method is demonstrated with examples: one road sign problem with a mix of two wall-following behaviors, and a more complex sequenced light-avoiding cockroach behavior. The results show that association rules are a powerful and practical way to implement rule-based controllers for reactive and semi-reactive robots.

1 Introduction

The work reported in this paper addresses the well-studied problem of making robots learn reactive behaviors from demonstrations. In general, the process is divided into three steps: 1. The robot is controlled, either by remote control by a human operator, or by a manually coded software controller to perform a certain task. The sensor data $S(t)$ at time 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. 3. The controller B is implemented in the robot, which hopefully manages to perform the demonstrated task autonomously. The various approaches to this general setup can be distinguished by the machine-learning technique chosen to arrive at the control law.

Reinforcement learning (RL) is a commonly used methodology (Lin (1991); Carreras et al. (2002)), which maps the state of the environment to an action that in turn maximizes the accumulated future rewards. The main advantage of RL is that it does not require all data to be available at the same time, as do most other machine-learning techniques. As a result, RL is suitable for online robot learning. The main disadvantages are long learning time, and problems with continuous variables (Carreras et al. (2002)). Artificial neural nets that have also been applied to the problem of finding reactive behaviors from data. Martin and Nehmzow (1995) use simple single layer perceptrons to represent behaviors for obstacle avoidance, wall following, cleaning, and route learning. Fuzzy rule bases have also been widely chosen to represent learned reactive behaviors. Ward et al. (2000) uses training data for a remote controlled robot to generate a fuzzy rule base capable of reproducing behaviors, such as wall following, corridor following, and docking. Evolutionary techniques have been combined with fuzzy rule bases to find optimal rules, e.g. in Hoffmann and Pfister (1997). In our approach, B is represented by a rule base with association rules. Association rules (Agrawal et al. (1993)) have been successfully used for data mining, where the goal is to explore complex databases to find patterns that might prove useful for various purposes. However, association rules have so far not been extensively used in robotics. One advantage with this machine-learning technique is the handling of uneven distribution of training examples. Most other techniques have a tendency to focus on the most common examples, and learn less from the scarce examples. For example, this is a well known problem when using neural nets for the learning process (Ward et al. (2000)).

The concept of association rules and how they are used to represent reactive behaviors is introduced in Section 2. The general method for building a con-

troller is described in Section 3. Results of practical experiments are presented in Section 4, including a road-sign-following behavior and a more complex cockroach-hide behavior. Section 5 concludes the paper with a summary and conclusions.

2 Behaviors as Association Rules

Association rules are a way of expressing dependencies between items in databases. 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 on how many observations of X and Y the computation of *strength* is based. For more information about these and related measures see Hellström (2003a).

2.1 Behavior Representation

The robotics framework in this paper is basically reactive, and each behavior 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 is 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

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

In our experiment we have a Khepera robot with 8 infrared sensors $IR_0, IR_1, .., IR_7$ to measure the 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.). For experiment 1, each sensor readout IR_i is split into 3 ranges $0, 1, 2$. For experiment 2, 4 ranges are used and represented by a discrete variable ir_i according to

$$ir_i = \begin{cases} 0 \text{ if } & 0 \leq IR_i < 100 & \text{(long distance)} \\ 1 \text{ if } & 100 \leq IR_i < 600 & \text{(medium dist.)} \\ 2 \text{ if } & 600 \leq IR_i < 900 & \text{(short dist.)} \\ 3 \text{ if } & 900 \leq IR_i \leq 1023 & \text{(very short distance)} \end{cases} \quad (2)$$

The robot has two wheels with independent motor control, so 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*
9	0	0	stop
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:

if $0 \leq IR_1 < 100 \wedge 100 \leq IR_2 < 600$ 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 Building a Controller

The rule base to control the robot is generated from data recorded from a manually programmed controller, demonstrating the required behavior. In this way we obtain a set of stimuli/response pairs that can be used to automatically generate a rule base. This rule base then replaces the controller, and hopefully produces 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. Algorithms that efficiently search large databases for association rules have been previously developed (e.g. Agrawal et al. (1993)).

The generated rules are implemented as a controller in the robot. 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. The right-hand side $y = a$ of the rule is used to control the robot. 2. More than one rule fires. The one with the highest *strength* is chosen. 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: to which rule does the sample belong? We have successfully designed and implemented a method called *k-nearest rules*, based on the classification technique k-nearest neighbors (kNN). For more information, refer to Hellström (2003a).

4 Experiments

We present results from two experiments demonstrating the power of using association rules to model reactive behaviors in the way described in the previous section. The experiments also show how non-reactive behaviors can be tweaked into the reactive framework by pre-processing the sensor data.

4.1 Experiment 1

This experiment deals with the Road sign problem (Linåker and Jacobsson (2001)), in which the robot has to act on a road sign it had 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 1.

Our approach is to let the robot act on preprocessed sensor data with a perceptual decay (Werger (1999)). The perception of a road sign remains even after the stimuli have 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 idea with perceptual decay is illustrated in Figure 2. The original stimuli as a function of time are shown in the lowermost pane. The perceptual decay in the middle pane shows how the perception remains and gradually decays after the original stimuli has disappeared. The uppermost pane shows another processing of the stimuli used in experiment 2.

The demonstrated 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 sign, 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 sensors for left and right bulb detection are denoted AL_l and AL_r respectively. To enable the robot to act on a road sign that appears and disappears before a junction, a virtual road sign sensor RS is defined as:

$$RS = \begin{cases} 2 \text{ if } & decay(AL_l) > decay(AL_r) \\ 0 & otherwise \end{cases} . \tag{5}$$

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 the robot's sight. 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 is added to the 8 infrared sen-

sors $ir_0, ir_1, .., ir_7$ as an additional input, and serves 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 pseudo code below:

```
if      RS = 2
        left-wall follower
else                                    (6)
        right-wall follower
end
```

where *left-wall follower* and *right-wall follower* are simple rule-based controllers described in Hellström (2003b). In step 2 of the basic learning process (see Section 1,) 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 in two categories: left-wall following and right-wall following. Of course, rules common to both behaviors may 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 sensor data is discretized in 3 ranges, and the actions in 8 categories as described in (3) (the stop action is never used in this example.) The training data is then used to automatically generate association rules, which in turn are used to construct a robot controller.

Table 1 shows performance for a number of different controllers with different numbers of rules. The number of rules is set by giving a lower limit to the *strength* value. Each controller is evaluated on one row in the table. The rules are applied to two data sets, the 1000 samples big training data set, which was used to generate the rules, and a test data set separately generated. The e_{tr} and e_{te} are the fractions of samples that give incorrect action when compared to the manually programmed controller. By demanding a *strength* value equal to 1.0, 31 rules are selected. The column labeled $0rule\%$ is the fraction of samples, for which no matching rule can be found in the controller's database. The 31 rule controller leaves 9.6 % of the samples not matched by any rule. The 1-nearest rule developed in Hellström (2003a) handles this reasonably well with 5.0 % incorrect actions on the test data set. The column labeled $1rule\%$ is the fraction of samples covered by exactly one rule. The rightmost 3 columns are the fractions of samples covered by 2, 3 and more than 3 rules respectively. 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 by a controller with the 43 rules with *strength*$\geqslant 0.95$. These rules give minimum error on both training and test data sets. 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 (Freitas (2000)), while it is very common in other areas of machine learning. However, acting on rules with very low *strength* or *support* corresponds to adding more nodes to a neural net, 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 using association rules for prediction or induction, in particular with noisy data, such as robot applications.

Table 2 lists a few of the generated rules and shows that not all rules responsible for turning 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 the rules are 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. For a more detailed analysis of the road sign experiment, see Hellström (2003a) and Hellström (2003b).

4.2 Experiment 2

This experiment aims at developing a rule base capable of mimicking the behavior of an imagined light-avoiding cockroach. A program performing the following robot behaviors is first developed (refer to Figure 3):

- If the light is switched off, explore the surroundings while avoiding obstacles
- If the light is switched on, perform the following sequence: 1. Turn around 180 degrees. 2. Move in a straight line to a wall. 3. Follow the wall until a hiding place is found. 4. Turn around and stop until the light is switched off.

This is a fairly challenging task even for a human robot programmer. In reality it took many days to

construct a program able to successfully perform all the described steps with the Khepera robot. It is clear that a pure reactive approach is not enough to achieve step 1 above. For this reason pre-processing of the ambient light sensor is introduced. The layout of this "habituation" function is illustrated in the topmost pane of Figure 2. The initial response follows the actual stimuli (the ambient light sensor) but falls off after a fixed time. In this way it is possible to model a time limited response in a semi-reactive fashion. The actual behavior is purely reactive but the pre-processing is not. It should be noted that the time for the response to fall off is tailored to match the time it takes for the robot to turn 180 degrees, i.e. to complete sub-behavior 1 above. This is necessary to make a reactive modelling possible, but may at first look like cheating. However, both animals and humans exhibit such tailored perception for various behavioral support. And after all, already the choice of sensors for a robot dictates which behaviors are feasible for the robot. The rest of the behavior described above can be programmed in a purely reactive fashion, using the infrared sensors to identify a hiding place and to turn around. The programmed controller is run with a cycle time of 0.1 seconds for 1000 seconds. This results in 10000 samples of training data. The data is then used to generate a rule base, which manages to reproduce the entire cockroach-like behavior using 100 association rules. In particular, the method manages to reproduce the time-limited turn in step 1 by including the pre-processed ambient-light sensor in the rules controlling the rotation.

5 Summary

We have demonstrated how association rules can be used by intelligent robot controllers for learning reactive and semi reactive behaviors. Two techniques to extend the reactive paradigm in this context have been presented. One road-sign-following task uses perceptual decay to achieve a memory of the type of the latest road sign. Another pre-processing of sensor data introduces habituation and makes it possible to implement a sequenced light-avoiding cockroach behavior.

References

Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216,1993.

M. Carreras, P. Ridao, J. Batlle, and T. Nicosevici. Efficient learning of reactive robot behaviors with a neural-q learning approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

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

T. Hellström. Association Rules for Learning Behavioral Mappings in Robotics. Technical Report UMINF-03.12 ISSN-0348-0542, Department of Computing Science Umeå University, Umeå Sweden, 2003a.

T. Hellström. Learning robotic behaviors with association rules. In Nikos Mastorakis, editor, *WSEAS transactions on systems*, 2003b. ISBN 1109-2777.

F. Hoffmann and G. Pfister. Evolutionary design of a fuzzy knowledge base for a mobile robot. *Int. J. of Approximate Reasoning*, 17(4):447–469, 1997.

Long Ji Lin. Programming robots using reinforcement learning and teaching. In *AAAI*, pages 781–786, 1991.

Fredrik Linåker and Henrik 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.

Paul Martin and Ulrich Nehmzow. Programming by teaching: Neural network control in the manchester mobile robot. In *Proceedings Intelligent Autonomous Vehicles*, 1995.

Koren Ward, Alexander Zelinsky, and Phillip McKerrow. Learning robot behaviours by extracting fuzzy rules from demonstrated actions. *Australian Journal of Intelligent Information Processing Systems*, 2000.

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

Table 1: 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

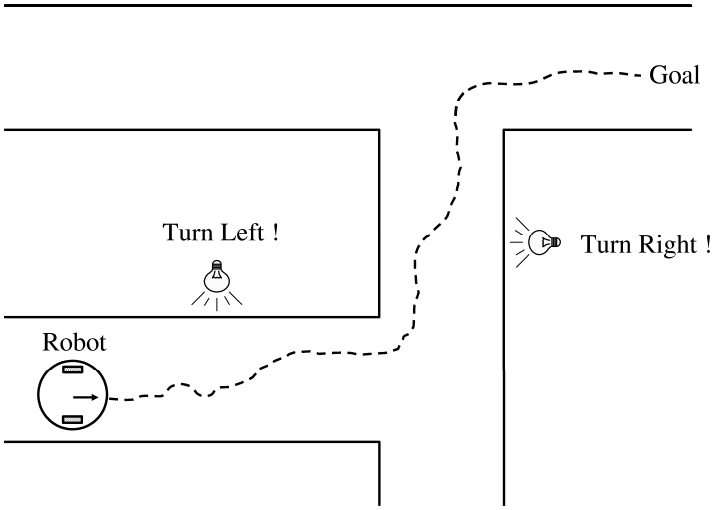

Figure 1: The road sign problem, adapted from Linåker and Jacobsson (2001), in which the robot has to decide on a left or right turn in each junction, depending on the past stimulus from the road signs. Our approach is to add a perceptual decay to the road sign perception. The robot switches between a left- and right-wall following behavior to perform the turnings in the crossings.

Table 2: Part of generated rule base for road sign controller. The binary RS variable controls left- and right-wall following.

Rule	**No.**	**Coverage**	**Support**	**Strength**
$ir_1 = 2 \wedge ir_2 = 2 \Rightarrow y = -3$	1	4	4	1.00
$ir_0 = 1 \wedge ir_1 = 2 \Rightarrow y = -3$	2	4	4	1.00
$ir_1 = 2 \wedge RS = 2 \Rightarrow y = -3$	3	6	6	1.00
$ir_0 = 1 \wedge ir_2 = 1 \wedge ir_6 = 1 \Rightarrow y = -3$	4	3	3	1.00
$ir_1 = 1 \wedge RS = 2 \Rightarrow y = -3$	5	67	67	1.00
$ir_2 = 2 \wedge ir_3 = 0 \Rightarrow y = -3$	6	2	2	1.00
$ir_2 = 1 \wedge ir_6 = 2 \wedge ir_7 = 1 \Rightarrow y = 3$	7	2	2	1.00
$ir_4 = 2 \wedge ir_6 = 1 \Rightarrow y = 3$	8	7	7	1.00
$ir_4 = 2 \wedge ir_7 = 1 \Rightarrow y = 3$	9	2	2	1.00
$ir_2 = 1 \wedge ir_4 = 1 \Rightarrow y = 3$	10	26	26	1.00
$ir_1 = 1 \wedge ir_4 = 2 \Rightarrow y = 3$	11	4	4	1.00
$ir_0 = 1 \wedge ir_4 = 2 \Rightarrow y = 3$	12	2	2	1.00
$ir_4 = 2 \wedge RS = 0 \Rightarrow y = 3$	13	25	25	1.00
$ir_3 = 2 \wedge RS = 0 \Rightarrow y = 3$	14	32	32	1.00
$ir_0 = 2 \wedge ir_1 = 0 \wedge RS = 2 \Rightarrow y = 0$	15	94	94	1.00
$ir_4 = 0 \wedge ir_5 = 2 \wedge RS = 0 \Rightarrow y = 0$	16	148	148	1.00
$ir_1 = 2 \wedge RS = 0 \Rightarrow y = -1$	17	8	8	1.00
$ir_1 = 2 \wedge ir_2 = 0 \Rightarrow y = -1$	18	6	6	1.00
$ir_0 = 1 \wedge ir_5 = 1 \wedge ir_6 = 1 \Rightarrow y = -1$	19	3	3	1.00
$ir_0 = 1 \wedge ir_6 = 1 \wedge RS = 0 \Rightarrow y = -1$	20	3	3	1.00
$ir_0 = 1 \wedge ir_7 = 1 \wedge RS = 0 \Rightarrow y = -1$	21	7	7	1.00
$ir_0 = 2 \wedge RS = 0 \Rightarrow y = -1$	22	13	13	1.00
$ir_3 = 0 \wedge ir_5 = 1 \wedge ir_6 = 0 \wedge RS = 0 \Rightarrow y = -1$	23	116	116	1.00
$ir_4 = 0 \wedge ir_5 = 1 \wedge RS = 0 \Rightarrow y = -1$	24	142	142	1.00
$ir_4 = 1 \wedge RS = 2 \Rightarrow y = 1$	25	13	13	1.00
$ir_1 = 0 \wedge ir_2 = 2 \wedge RS = 2 \Rightarrow y = 1$	26	101	101	1.00

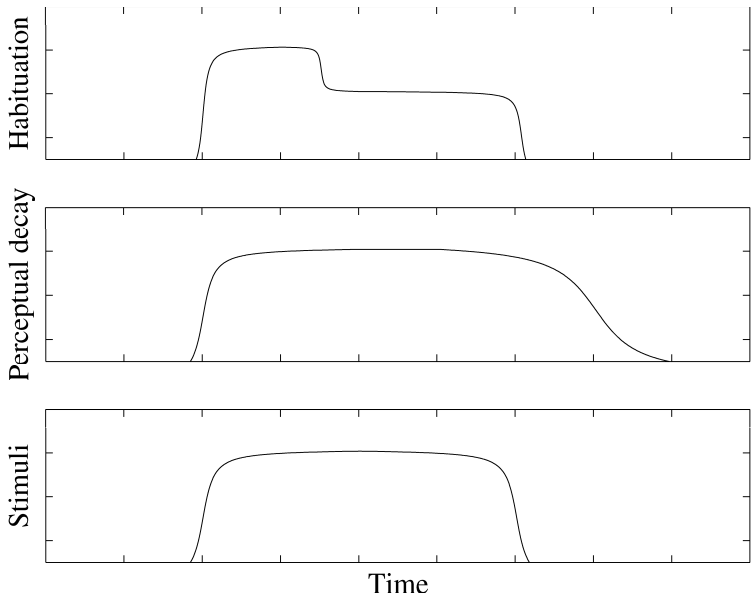

Figure 2: Two ways of introducing non reactivity by pre-processing of sensor data. The perceptual decay enables extended response to a stimulus. The habituation enables a sequence of two behaviors as a response to a stimulus.

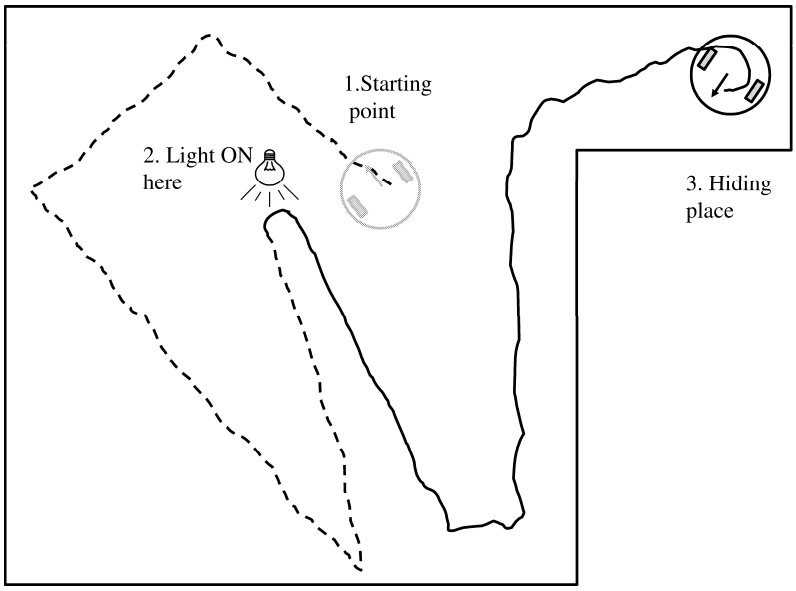

Figure 3: A robot emulating a cockroach's light-avoiding behavior. Between 1. and 2. the light is off and the robot moves around randomly, while avoiding obstacles. At 2 the light is switched on and the robot turrns 180 degrees, moves until it hits a wall, which it follows until it reaches a hiding place, where it turns around and stops.