Mobile Robot Learning of Delayed Response Tasks through Event Extraction: A Solution to the Road Sign Problem and Beyond

Fredrik Linåker†,‡ **Henrik Jacobsson**†,‡
†Department of Computer Science, University of Skövde, Sweden
‡Department of Computer Science, University of Sheffield, United Kingdom
{fredrik.linaker, henrik.jacobsson}@ida.his.se

Abstract

We show how event extraction can be used for handling delayed response tasks with arbitrary delay periods between the stimulus and the cue for response. Our approach is based on a number of information processing levels, where the lowest level works on raw time-stepped based sensory data. This data is classified using an unsupervised clustering mechanism. The second level works on this classified data, but still on the individual time-step basis. An event extraction mechanism detects and signals transitions between classes; this forms the basis for the third level. As this level only is updated when events occur, it is independent of the time-scale of the lower level interaction. We also sketch how an event filtering mechanism could be constructed which discards irrelevant data from the event stream. Such a mechanism would output a fourth level representation which could be used for delayed response tasks where irrelevant, or distracting, events could occur during the delay.

1 Introduction

Consider an automated robot driver that navigates the streets to reach a certain goal location. On its journey, it encounters road signs, describing the upcoming junctions. Having detected a road sign, the system needs to be able to later on make the appropriate decision based on this information. That is, based on a stimulus, the system needs to store information and then make an appropriate response once the junction has been reached. This may not occur for several seconds or even minutes, depending on the vehicle's traveling speed and the distances involved. This problem was described in [Rylatt and Czarnecki, 2000], which in turn was based on a more abstract description by [Ulbricht, 1996].

The problem is in fact of a very general nature, in that it involves a *delayed response task* (Figure 1). Such tasks are quite common in real-life, involving associations between inputs and actions at different points in time. As shown by [Rylatt and Czarnecki, 2000], most existing neural network approaches are however quite inept at handling these sorts of problems. Rylatt and Czarnecki showed that, in fact, even their own, for the task specially constructed, recurrent neural network architecture was unable to learn the appropriate associations if they lay more than just a few time-steps apart.

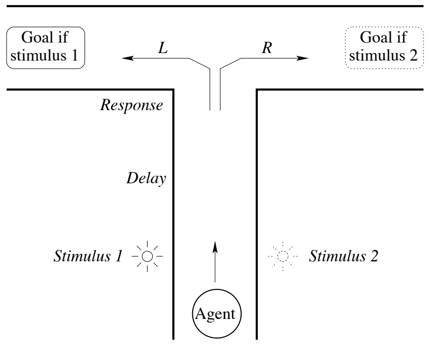


Figure 1: The delayed response task, adapted from [Ulbricht, 1996]. The robot travels past a stimulus, here a light either on the left or the right side, then continues down the corridor for a number of steps until it reaches the junction at which time the robot needs to decide whether it should turn left or right, depending on the location of the light it passed earlier.

In this paper, we present a quite different approach from Rylatt and Czarnecki, in that we do not work directly on the input sequence but instead let an unsupervised system extract a set of *events* from the inputs and then we work on this sequence of events. As we will show, the time intervals between events may in fact be arbitrarily large without affecting the performance of the system; a drastic change from previous approaches. We also describe a more complex situation, the 'Extended Road Sign Problem', which involves having distractions occur during the delay period. Our solution is based on having several information processing levels, as depicted in Figure 2.

Level 1: Contains raw multi-dimensional sensor data, which is time-step based[1]. This is the level where [Rylatt and Czarnecki, 2000] approached the delayed response task. We however argue, like [Nolfi and Tani, 1999], that real-world

[1]Where a time-step is defined as a single update of sensors, neuronal elements, and actuators with a regular time interval, typically in the millisecond range.

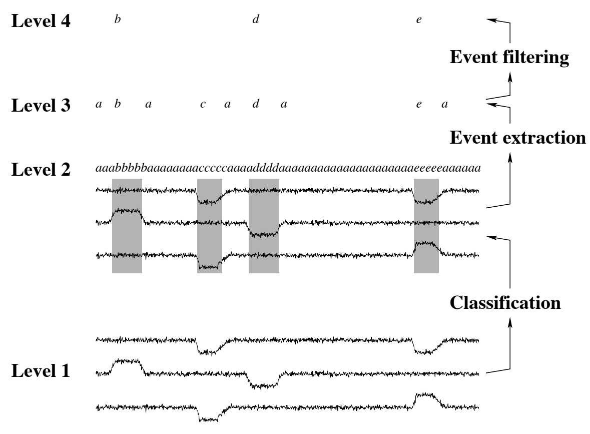


Figure 2: The proposed four-level information processing architecture which can handle delayed response tasks with very long-term dependencies.

task dependencies do most often not manifest themselves at this level of individual time-stepped neuronal updates, but rather on much slower time scales, involving several seconds or even minutes. Rylatt and Czarnecki's somewhat simplified simulations and specially modified neural network architecture was only able to learn dependencies which lay up to 13 time-steps apart. But as most robot systems have a high sampling frequency, Rylatt and Czarnecki's system would not be sufficient. For example, the Khepera robot we use in our experiments, has sensor sampling rates of approximately 20 times per second. Rylatt and Czarnecki's system would therefore, in effect, not be able to learn tasks involving even just single second delays. In the following, we will show that by using event extraction, the delays can instead be arbitrarily large and this enables the system to handle more realistic long-term dependencies.

Classification: The process whereby the raw multi-dimensional sensor data is divided into a set of classes. While [Nehmzow and Smithers, 1991] employed a set of manually pre-defined (fixed) classes for this, [Tani and Nolfi, 1998] instead let the system determine the class structure by itself, thereby reducing the user intervention. However, Tani and Nolfi still had to manually specify the number of classes, and had to divide the training into several different phases. They also had large problems with inputs which were distinct but not very frequent in the training set, and the learning process was very slow. Recently, [Linåker and Niklasson, 2000] have constructed a more flexible classification system, the ARAVQ, which is able to swiftly classify inputs into a dynamic number of automatically extracted classes, overcoming most of the problems in Tani and Nolfi's system. (The ARAVQ system is the one used in the following simulations.)

Level 2: Contains raw time-step based multi-dimensional data which has been tagged with class labels. If each class is alloted a character in the alphabet, the input can be re-written, with some loss of information, as a (very long) letter sequence. This is the level at which [Ulbricht, 1996] approached the road sign problem, trying to learn associations between the letters in the sequence. She did not, however, provide any account for how the input had become this letter sequence (i.e. no Level 1 nor any classification), thereby working on essentially *ungrounded* symbols. And, further, as her system was still time-step based, she had the same difficulties learning long-term dependencies as Rylatt and Czarnecki had in their Level 1 system.

Event extraction: The process whereby only the transitions between class membership of the lower level data are extracted, thereby filtering out repetitions. This is a fairly straightforward mechanism as long as the class membership is exclusive (each input belonging to one—and only one—class); if two succeeding inputs are classified differently, an event is generated. The detection of an event generates a signal to the next level.

Level 3: Contains general events, interspersed over long periods of time. Updates occur on a considerably slower time-scale than the time-step based levels below. This means that longer time-dependencies can be detected, as noted by [Tani and Nolfi, 1998]. While Tani and Nolfi's robot system worked at this level, it did not involve any coupling back to the real-world as the input was merely classified and filtered, and not acted upon in any manner. That is, their system did not use the extracted events to control the robot; it was only an idle observer of what was going on. We here provide an account for how extracted events can be used to learn delayed response tasks and also how this can be used to identify candidates which should pass through an event filtering on to yet another level.

Event filtering: The process which discards events which are considered as irrelevant for accomplishing the task. This process requires that the events have been rated with some sort of 'usefulness' score, related to how relevant they are for achieving the task. This event rating should ideally be based on a delayed reinforcement learning system, such as Q-learning, as rewards in the real world do often not come immediately as an action is performed. In the following we provide a simpler but less realistic evaluation mechanism, based on a recurrent neural network which has learnt a simple supervised version of the task at Level 3 (see Section 4). The idea behind this is that once a simple version of the task has been learnt on a low level, it can be generalized to more complex situations on higher information processing levels that have access to longer time horizons.

Level 4: Contains only the events which are considered as relevant for achieving the task. It is updated on an even slower time-scale than Level 3 and thus can handle events that have occurred even further apart.

It is worth noting that from Level 2 and upwards, the system can work on an essentially symbolic representation of the input sequence. These symbols provide a representation whose size is virtually independent from the dimensionality of the actual sensory and motor systems. This relaxes the information processing and storage demands on the system, assuming that the symbolic representation is more compact than its corresponding input, which usually is the case.

The next section describes an example of how a simple system can be constructed, based on straightforward building blocks, in order to achieve the discussed information processing.

2 Architecture

In addition to the information processing capabilities described in the previous section, the system needs to be able to control the robot, i.e. making the appropriate response once the cue for responding comes. There are several possibilities of executing actions. Levels 1 and 2 are both time-step based, which means that the inputs can be directly coupled to a single action which lasts a proportionate single time-step. This can accommodate for simple, non-goal-oriented, and/or 'innate' reflex actions.

However, associations between inputs at Levels 3 and upwards are based on events. These events can occur at widely interspersed points in time. A single time-step action is therefore not appropriate as a response; the response should ideally affect the performance the entire time interval up to the next event. A simple solution is to repeatedly execute the *same* action until the next event occurs, e.g. to keep turning in one direction until the next event occurs. This would however be a very rigid and inflexible solution, not allowing the system to modify its responses into smoother real-time interactions.

Instead, we propose, that the event-based levels affect, or modulate, the actual input-to-output (sensation-to-action) *mapping* of the time-step based levels. This modulation would also give the system the ability to focus on particular sensor subsets which are of most importance to the particular response. We show this by manually constructing a set of very simple input-to-output mappings, or *behaviours*, which the event-based levels can choose between. Each behaviour only works on a subset of the available sensory channels. As the higher levels themselves do not act directly on the actuators, there is also no need for between-level action selection, something which has caused problems in other layered control architectures such as Brooks' well-known Subsumption Architecture. Our architecture is summarized in Figure 3.

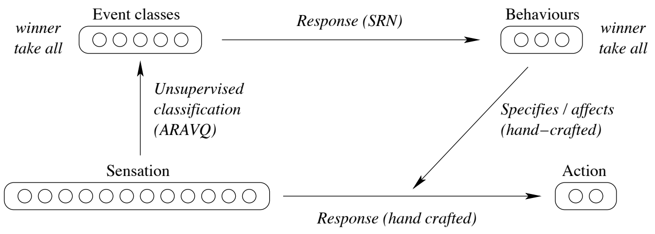


Figure 3: The architecture realizing information processing levels 1 through 3 in Figure 2, including the means for controlling the agent's actions.

Inputs from the robot sensors were fed through an ARAVQ network (Section 2.1), which classified the inputs into a set of classes. When the classification became different for two succeeding time-steps, an event was generated and a localistic representation of the winner was fed into a Simple Recurrent Network (SRN, see Section 2.2), which learnt associations between inputs (events) and outputs (behaviours) at different time points. The SRN specified which of the behaviours (Section 2.3) the robot was to employ until the next event occurred.

2.1 The ARAVQ

The adaptive resource allocating vector quantization (ARAVQ) network [Linåker and Niklasson, 2000] is a vector quantization network which contains a set of model vectors, representing event classes. When a series of novel and stable inputs are encountered, the system dynamically incorporates additional model vectors. The number of allocated model vectors is determined by the characteristics of the input signal, which in turn reflects the characteristics of the environment, or the agent-environment interaction, that underlies the sensory flow which we apply the ARAVQ network to here. It is also biased by the ARAVQ's parameter settings.

The ARAVQ has four user-defined parameters: a novelty criterion δ, a stability criterion ϵ, an input buffer size n and a learning rate α. These are all explained below. In order to cope with noisy inputs, the ARAVQ filters the input signal using the last n input vectors, which are stored in an input buffer $X(t)$. The values in the input buffer are averaged to create a more reliable, *filtered*, input $\overline{x}(t)$ to the rest of the network. That is, a finite moving average $\overline{x}(t)$ is calculated for the last n time-steps.

There is a set $M(t)$ of model vectors (each one representing an event class), which is initially empty. (The ARAVQ does not start working until the input buffer is filled, i.e, until time-step $n-1$.) Additional model vectors are only allocated when novel and stable inputs are encountered, i.e., when the following criteria are fulfilled:

- The input is considered as *novel* if the Euclidean distance between the existing model vectors $M(t)$ and the last n inputs, compared to the distance between the moving average $\overline{x}(t)$ and the last n inputs is larger than the distance δ.
- The input is considered *stable* if the difference between the actual inputs $x(t), x(t-1), \ldots, x(t-n+1)$ and the moving average $\overline{x}(t)$ is below the threshold ϵ.

For convenience, we define the following general distance measure between a set of (model/filtered input) vectors V and a set of actual inputs X:

$$d(V,X) = \frac{1}{|X|}\sum_{i=1}^{|X|} \min_{1\leq j\leq |V|}\{||x_i - v_j||\}; x_i \in X, v_j \in V, \tag{1}$$

where $||.||$ denotes the Euclidean distance measure. The distance between the filtered input and the actual inputs is defined as:

$$d_{\overline{x}(t)} = d(\{\overline{x}(t)\}, X(t)), \tag{2}$$

and the distance between the existing model vectors and the actual inputs is:

$$d_{M(t)} = \begin{cases} d(M(t), X(t)) & |M(t)| > 0 \\ \epsilon + \delta & \text{otherwise.} \end{cases} \tag{3}$$

Event Class Incorporation: If both the stability and novelty criteria are met, the filtered input is incorporated as an additional model vector:

$$M(t+1) = \begin{cases} M(t) \cup \overline{x}(t) & d_{\overline{x}(t)} \leq min(\epsilon, d_{M(t)} - \delta) \\ M(t) & \text{otherwise.} \end{cases} \tag{4}$$

Classification: Each time-step, a winning model vector $win(t)$ is selected, indicating which class the (filtered) input currently matches:

$$win(t) = \arg \min_{1 \le j \le |M(t)|} \{||\overline{x}(t) - m_j||\}; m_j \in M(t). \quad (5)$$

Adaptation: If the winning model vector matches the (filtered) input very closely, the (filtered) input is considered to represent a 'typical' instance of the class, and the model vector is modified to match the input even closer:

$$\Delta m_{win(t)} = \begin{cases} \alpha[\overline{x}(t) - m_{win(t)}] & ||\overline{x}(t) - m_{win(t)}|| < \frac{\epsilon}{2} \\ 0 & \text{otherwise,} \end{cases} \quad (6)$$

where α is a user-defined learning rate. The structure of the ARAVQ network is depicted in Figure 4.

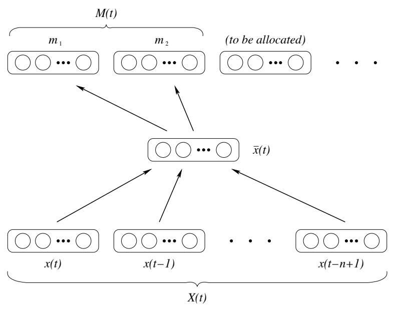


Figure 4: The ARAVQ network. The n last inputs are buffered and used to calculate a filtered input $\overline{x}(t)$. This particular network has allocated two model vectors, m_1 and m_2; additional model vectors will be allocated automatically when novel and stable inputs are encountered. (In the following, model vectors are for convenience labeled a, b, c, etc. instead of m_1, m_2, m_3, etc.)

2.2 The SRN

The simple recurrent network (SRN) [Elman, 1990] is essentially a three-layer feed-forward network which stores a copy of the previous hidden activation pattern and feeds it as additional input at the next time-step. This provides a memory trace of previous inputs which enables the network to learn associations over several time-steps.

The output of the SRN (for an arbitrary number of nodes) is defined as:

$$o_k(t) = f(\sum_j w_{kj} h_j(t) + w_{k\theta}) \quad (7)$$

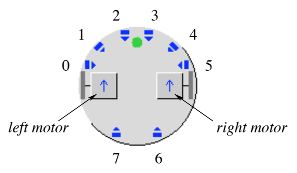
where the hidden activation is defined as

$$h_j(t) = f(\sum_i v_{ji} i_i(t) + \sum_m v_{jm} h_m(t-1) + v_{j\theta}) \quad (8)$$

and $i(t)$ is the input at time t. The index k is used for identifying the output nodes, j and m are used for the hidden nodes (at time t and $t-1$ respectively), and i is used for the input. Biases are introduced as additional elements in the weight matrices, w and v (indexed with θ). The function, f, is the sigmoid activation function $f(x) = 1/(1+e^{-x})$.

2.3 Behaviours

Three different input-to-output mappings, or behaviours, were constructed: a corridor follower, a left wall follower, and a right wall follower. Each behaviour only needed to use a subset of the available sensor readings; see Figure 5. The Khepera robot which was used has eight infrared proximity sensors with integer activation in the range $[0, 1023]$, 0 denoting no object present within sensor range and 1023 denoting an obstacle very close. The robot has two separately controlled wheels which can be set to integer values in the range $[-10, 10]$, -10 denoting maximum backward spinning, 0 no wheel movement, up to 10 which rotates the wheel forward at maximum speed.



(a)

```
/* Corridor follower */
if (sensor[4] > 800) {
  motor[LEFT] = 0;
  motor[RIGHT] = 5;
} else if (sensor[1] > 800) {
  motor[LEFT] = 5;
  motor[RIGHT] = 0;
} else {
  motor[LEFT] = 5;
  motor[RIGHT] = 5;
}
```

(b)

```
/* Left wall follower */
if (sensor[1] > 200) {
  motor[LEFT] = 5;
  motor[RIGHT] = -4;
} else if (sensor[0] < 800) {
  motor[LEFT] = 2;
  motor[RIGHT] = 5;
} else {
  motor[LEFT] = 5;
  motor[RIGHT] = 5;
}
```

(c)

```
/* Right wall follower */
if (sensor[4] > 200) {
  motor[LEFT] = -4;
  motor[RIGHT] = 5;
} else if (sensor[5] < 800) {
  motor[LEFT] = 5;
  motor[RIGHT] = 2;
} else {
  motor[LEFT] = 5;
  motor[RIGHT] = 5;
}
```

(d)

Figure 5: The Khepera robot and the hand crafted behaviours.

In addition to the three hand crafted behaviours, a simple selection mechanism was implemented. At each time-step, it detected the most active output node of the SRN and executed the code associated with the behaviour.

3 Experiments

A simulated version of the Khepera robot was used in the experiments. The activation of the eight distance sensors, the two motors, and two of the robot's light sensors, placed in concert with distance sensors 0 and 5 in Figure 5(a), were normalized to the range $[0.0, 1.0]$ and fed as input to the architecture. That is, the ARAVQ network received a total of 12 inputs. The parameter settings of the ARAVQ were $\delta = 0.7$, $\epsilon = 0.2$, $n = 10$ and $\alpha = 0.05$. This led to the extraction of eight different event classes for the constructed T-maze environments shown in Figures 6 and 7; each event class was allocated a separate shade and at each location, the winner of the classification process was plotted, leaving the trail shown in the figure.

The event extraction discards the repetitions of each class, leaving sequences which are only six characters long. A character was manually assigned to each of the eight extracted classes, for presentational purposes. An interpretation of each

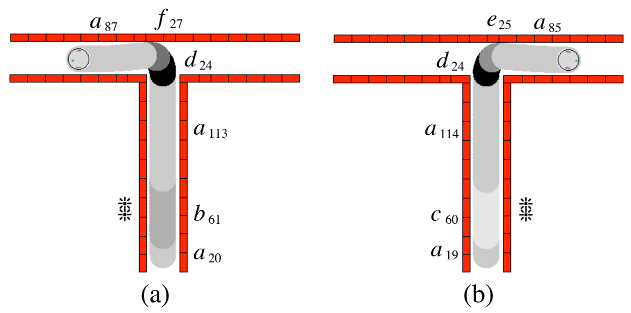


Figure 6: The two different cases for the delayed response task and the resulting input classification at each location along the simulated robot's path, the subscript denoting the number of repetitions of each class.

class is shown in Table 1, based on how the ARAVQ seems to apply the classes.

model vector	*interpretation*
a	corridor
b	corridor + left light
c	corridor + right light
d	junction
e	wall on left side only
f	wall on right side only
g	left-turning corner
h	right-turning corner

Table 1: The eight automatically extracted model vectors and how they can be interpreted.

Each of the three behaviours was also assigned a character, this time creating an 'output alphabet', as shown in Table 2.

behaviour	*description*
C	corridor following
L	left side wall following
R	right side wall following

Table 2: The three hand crafted behaviours.

The SRN worked on this eight-character input and three-character output alphabet. As two hidden nodes were used, a 8-input, 2-hidden, 3-output SRN was created. Network weights were randomly initialized in the interval $[-5.0, 5.0]$, the learning rate was 0.01 and a momentum of 0.8 was used. The training set was the two extracted sequences, for the left and right turn, respectively, as shown in Table 3. The network was trained for $10,000$ epochs using back-propagation through time (BPTT) which *unfolds* the recurrent connections of the network. The BPTT here unfolded the network 5 times.

The SRN had no problems learning the correct associations

case	*sequence*					
Left turn	a	b	a	d	f	a
	C	C	C	L	C	C
Right turn	a	c	a	d	e	a
	C	C	C	R	C	C

Table 3: The extracted sequences of model vector winners and the behaviours that should be selected for the paths taken in Figure 6.

between the light stimuli b and c and the behaviours L and R occurring two events later. That is, the system could turn in the right direction, irrespective of the length of the delay, as this information was removed in the event extraction. The Road Sign Problem was thus solved. We now, however, have a similar problem to that of [Rylatt and Czarnecki, 2000], but on the level of intermediate *events* instead of intermediate time-steps. We call this problem 'The Extended Road Sign Problem'.

4 The Extended Road Sign Problem

While the length of the delay between the stimulus and the response has become irrelevant through the use of event extraction, the system would still have problems handling distracting events during the delay. That is, if the input changes drastically during the delay, intermediate events will be generated. This means that the problem of finding relationships between the stimulus and the subsequent response will become harder as there are a number of distracting events which have taken place in between. Examples of this are shown in Figure 7 where there is a right or left turn in the corridor after the stimulus has been passed, generating another three events which however are of no relevance for the task, i.e. they serve only as distractions.

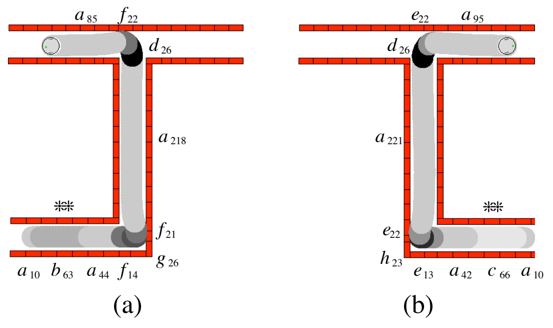


Figure 7: Two examples of distracting events (turns) happening in between the stimulus (light) and the cue (junction).

The SRN still managed to find the correct association, but only if it was first trained on the simpler tasks, and then continued training on the more difficult examples shown above. The hidden node activation plot of an SRN which has learnt the task is depicted in Figure 8. Note that a number of clusters have formed for each of the functional states the robot can be in.

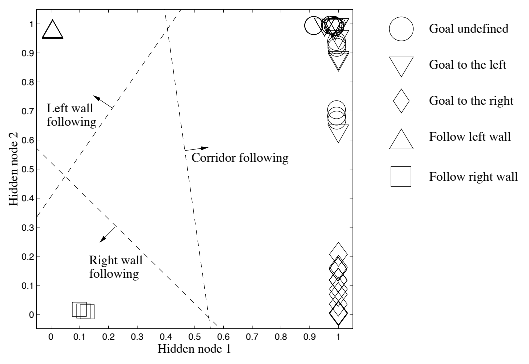


Figure 8: Hidden node activation of the SRN.

When irrelevant inputs, such as corners (input characters g and h), are received by the SRN, the state remains relatively stable, i.e. it stays in the same location as the previous time-step. That is, large movements in the internal (hidden node) activation space occur only when *functionally important* events occur. When, for instance, input character b (left stimulus) is received, the activation jumps to the upper right corner and stays there until the input character d (the cue for responding) arrives. At that time, the activation quickly jumps to the upper left corner, making the robot activate its left wall following behaviour, effectively starting to turn left at the junction. A similar situation occurs if instead the other stimulus is present, where the bottom right and left corners are used by the SRN. (Before either stimuli has been encountered, the SRN state is in the upper right corner, i.e. this particular robot has a bias for turning left at junctions.)

We can now also sketch how a solution to the Extended Road Sign Problem might look. Note that if the hidden activation space of this SRN was clustered, using for example another ARAVQ network, the functionally unimportant events would likely cause repetitions of the same class winner as they would lead to only small perturbations in the state space. Only when functionally important events occur, such as the stimulus or cue, do large jumps in activation space occur, thereby leading to another class perhaps becoming the best match. Then using the same filtering process used for repetitions of input patterns, the irrelevant events would be removed. Note that this solution is based on a Level 3 system (SRN) which has already successfully learnt the associations in a relatively simple scenario. The filtering can then extract information which effectively lets the next higher level (Level 4) handle delays with an *arbitrary* number of distracting events as they will be filtered out in the aforementioned process.

5 Conclusions

We have shown how a layered information processing system can be constructed which handles delayed response tasks like the Road Sign Problem [Rylatt and Czarnecki, 2000]. Our solution is based on attacking the problem at a higher level of abstraction than the raw time-step based sensory data. As shown, the learning system (in this case an SRN) has a considerably easier task when the redundant data has been filtered out, letting the system instead work on a sequence of discrete events. These events are grounded in sensori-motor interactions. As discussed, the event extraction provides the means for handling *arbitrarily* long delays between the stimulus and the subsequent cue for response.

In addition to handling the Road Sign Problem, we suggest an Extended Road Sign Problem. This involves distractions during the delay, thereby putting further demands on the system not to lose track of what it is supposed to do. As discussed, another abstraction level, which works on *filtered* event streams, could be added. This filtering would be task-specific and would be based on that the system has already learnt, on a simple version of the task, which of the inputs are relevant. We believe this approach will steer us in the right direction on the road to acquiring more complex and intelligent behaviours from our robotic friends.

Acknowledgments

This research was funded by a grant from the Foundation for Knowledge and Competence Development (1507/97), Sweden and the University of Skövde, Sweden.

References

[Elman, 1990] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[Linåker and Niklasson, 2000] F. Linåker and L. Niklasson. Time series segmentation using an adaptive resource allocating vector quantization network based on change detection. In *Proc. of the Int. Joint Conf. on Neural Networks*, volume VI, pages 323–328. IEEE Computer Society, 2000.

[Nehmzow and Smithers, 1991] U. Nehmzow and T. Smithers. Mapbuilding using self-organising networks in really useful robots. In *Proc. of the First Int. Conf. on Sim. of Adaptive Behavior*, pages 152–159. MIT Press, 1991.

[Nolfi and Tani, 1999] S. Nolfi and J. Tani. Extracting regularities in space and time through a cascade of prediction networks. *Connection Science*, 11(2):125–148, 1999.

[Rylatt and Czarnecki, 2000] R.M. Rylatt and C.A. Czarnecki. Embedding connectionist autonomous agents in time: The ‘road sign problem’. *Neural Processing Letters*, 12:145–158, 2000.

[Tani and Nolfi, 1998] J. Tani and S. Nolfi. Learning to perceive the world as articulated. In *Proc. of the Fifth Int. Conf. on Sim. of Adaptive Behavior*, pages 270–279. MIT Press, 1998.

[Ulbricht, 1996] C. Ulbricht. Handling time-warped sequences with neural networks. In *Proc. of the Fourth Int. Conf. on Sim. of Adaptive Behavior*, pages 180–189. MIT Press, 1996.