Reinforcement Learning in Partially Observable Mobile Robot Domains Using Unsupervised Event Extraction

Bram Bakker[1], Fredrik Linåker[2], Jürgen Schmidhuber[1]

[1] *IDSIA, Manno-Lugano, Switzerland, bram@idsia.ch/juergen@idsia.ch* *
[2] *Dept. of Computer Science, University of Skövde, Sweden, fredrik@ida.his.se*

Abstract

This paper describes how learning tasks in partially observable mobile robot domains can be solved by combining reinforcement learning with an unsupervised learning "event extraction" mechanism, called ARAVQ. ARAVQ transforms the robot's continuous, noisy, high-dimensional sensory input stream into a compact sequence of high-level events. The resulting hierarchical control system uses an LSTM recurrent neural network as the reinforcement learning component, which learns high-level actions in response to the history of high-level events. The high-level actions select low-level behaviors which take care of real-time motor control. Illustrative experiments based on a Khepera mobile robot simulator are presented.

1 Introduction

Reinforcement learning is an intuitively appealing approach for developing intelligent robots. It does not require a person to program the desired behavior by hand, or even show the robot desired actions in different situations. Instead, it potentially allows the robot to learn virtually autonomously, based only on scalar rewards for reaching particular goals.

Several difficulties stand in the way of widespread application, however. First, most reinforcement learning work is concerned with small, discrete environments, whereas a robot's environment is inherently large and continuous. Second, most reinforcement learning algorithms operate on the assumption of complete observability of the state (Markov Decision Processes, MDPs), whereas in many realistic robot domains the state is only partially observable (Partially Observable Markov Decision Processes, POMDPs). Partial observability corresponds to, for instance, different hallways looking the same, meaning that a mobile robot cannot rely on perception alone but must remember events from the past ("I just passed the men's room and took a left turn") to disambiguate the hallway. Related to both issues is the possibility of very long time lags, either between actions and finally achieving a goal, or between events that need to be remembered and actions that depend on them. Long time lags make temporal credit assignment very difficult: which of the many past actions are to be credited for good behavior, and which of the many past events need to be remembered?

This paper attempts to make some progress on these issues by combining different techniques. The main idea is to combine reinforcement learning with unsupervised learning. This idea was also suggested by Hinton [6] when he said about the brain:

> There is probably some basic way of unsupervised learning and once it constructed representations that way, there is probably some basic way of reinforcement learning based on these representations.

In this paper, the unsupervised learning method accomplishes both spatial and temporal abstraction (or compression) of the robot's sensory inputs. A high-level representation of the environment is thus obtained in which reinforcement learning becomes much more feasible than in the raw, low-level representation. The unsupervised learning method does not in itself solve the problem of partial observability, but it does make it easier. In the spatially and temporally compressed representation it is easier for the reinforcement learning system to find the temporal dependencies which can disambiguate the state. In this work, the reinforcement learning system is based on an LSTM recurrent neural network, which has been shown to be good at learning such temporal dependencies [7, 1].

The next section describes the overall system in more detail, and discusses related work. Section 3 presents results of experiments done using a Khepera robot simulator. Section 4, finally, contains a general discussion.

*The first author's work was partially funded by a CSEM robotics grant to Jürgen Schmidhuber.

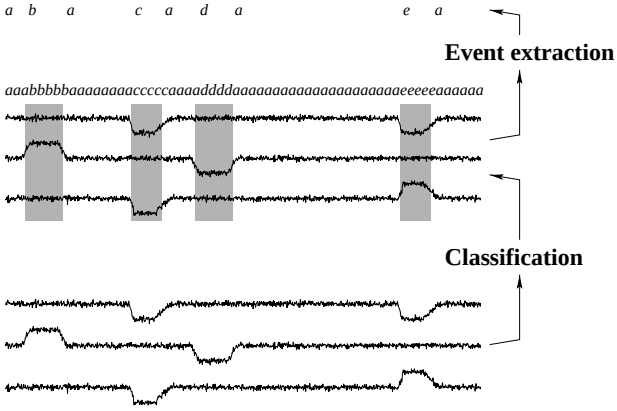

Figure 1: *A schematic representation of event extraction. The horizontal dimension represents time. First, the continuous sensor vector is classified. Next, only changes in classification are passed on as significant events.*

2 The learning system

2.1 Unsupervised event extraction

One component of the overall learning system is an unsupervised learning system doing *event extraction.* Mobile robots typically have many, noisy sensors, sampled at a high rate, e.g. 20 Hz. From this time-series of many noisy, high-dimensional data points, the event extraction mechanism extracts a compact sequence of high-level "events" or "concepts". Figure 1 shows the general idea. The mechanism accomplishing this is called an Adaptive Resource Allocation Vector Quantization (ARAVQ) network [9]. It basically matches incoming sensor vectors, i.e. the values coming from the different sensors, to stored model vectors in a manner similar to Kohonen maps, and it dynamically allocates new model vectors when it encounters novel and stable situations.

ARAVQ stores the last n input vectors in an input buffer. The values in the input buffer are averaged to create a more reliable, filtered input $\overline{x}(t)$ to the rest of the ARAVQ network. There is a set $M(t)$ of model vectors (each representing an event class), which is initially empty. The ARAVQ network operates in three stages:

Event class incorporation. Additional model vectors are only allocated when *stable* and *novel* inputs are encountered, i.e., when the following criteria are fulfilled. The input is considered stable if $d_{\overline{x}(t)}$, the average Euclidian distance between $\overline{x}(t)$ and the last n inputs, is below a threshold parameter ϵ. The input is considered novel if $d_{M(t)}$, the average Euclidean distance between the best matching stored model vector and the last n inputs, is larger than $d_{\overline{x}(t)}$ plus a distance parameter δ. If both of these criteria are met, the filtered input is incorporated as an additional model vector:

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

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

$$v(t) = \arg \min_{1 \leq j \leq |M(t)|} \{||\overline{x}(t) - m_j||\}; m_j \in M(t). \quad (2)$$

Only when the winning model vector changes, an event is "thrown" (as depicted in figure 1, top layer).

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 become even more like this input, in a manner similar to Kohonen maps:

$$\Delta m_{v(t)} = \begin{cases} \alpha[\overline{x}(t) - m_{v(t)}] & ||\overline{x}(t) - m_{v(t)}|| < \frac{\epsilon}{2} \\ 0 & \text{otherwise,} \end{cases} \quad (3)$$

where α is a learning rate parameter.

In summary, the ARAVQ network accomplishes two things that are good for reinforcement learning. First, it compresses and simplifies the high-dimensional sensor space and filters out much of the noise, producing clear-cut higher level concepts representing distinct, stable situations. Second, it compresses the long, high sampling rate timeseries into a short sequence of discrete events. In the present study, the ARAVQ learning phase precedes the reinforcement learning phase.

2.2 Reinforcement learning on the extracted concepts

The events or concepts coming out of the unsupervised learning system are the inputs to the reinforcement learning component. This component is a reinforcement learning Long Short-Term Memory (RL-LSTM) recurrent neural network [7, 1], whose basic architecture is depicted in figure 2. The input of the RL-LSTM network basically consists of the output units of the ARAVQ network. Thus, each input is a vector with one unit set to 1 (the winning concept) and the remaining ones set to 0.

The RL-LSTM network approximates the value function of the reinforcement learning algorithm called Advantage learning [4] with eligibility traces [1]. Each of the output units represents the Advantage value of one of the possible actions (see figure 2). The Advantage value function can be viewed as a transformation of the value function of the well-known

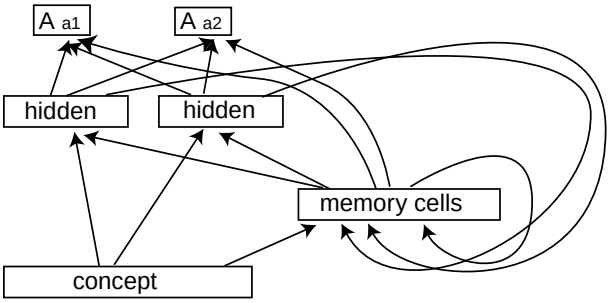

Figure 2: *Schematic architecture of the reinforcement learning LSTM network. The input consists of the winning concept extracted by ARAVQ. Each output unit represents the Advantage value of an action. The memory cells can store relevant information from the experienced history of concepts.*

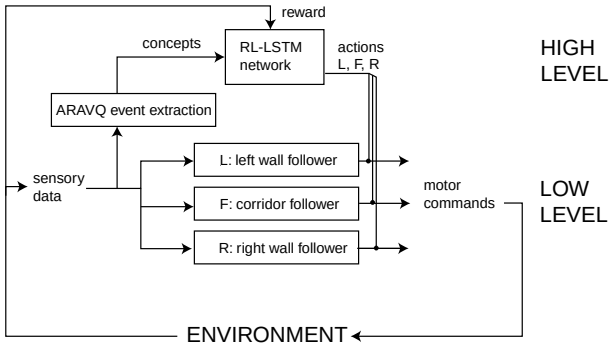

Figure 3: *The complete hierarchical control system. In response to events detected by ARAVQ, RL-LSTM produces high-level actions which select low-level behaviors.*

Q-learning algorithm. The true value of an environmental state is defined as

$$V^*(s) = \max_a A^*(s,a). \tag{4}$$

The optimal Advantage $A^*(s,a)$ for each action a in state s must satisfy the following equivalent of the Bellman equation:

$$A^*(s,a) = V^*(s) + \frac{\langle r + \gamma V^*(s') \rangle - V^*(s)}{\kappa} \tag{5}$$

where $\langle \cdot \rangle$ represents the expected value over all possible results of executing action a in state s, which leads to immediate reward r and a new state s'. γ is a factor discounting future rewards, and κ is a constant scaling the difference between optimal and suboptimal actions. The difference between the righthand side and the lefthand side of this equation, estimated at each point in time using the interaction with the environment, is a *temporal difference error*, which is propagated back through the network using LSTM's efficient variation of real-time recurrent learning (see [7, 1] for details). During learning various actions are tried out using the directed exploration technique described in [1]. Once the LSTM network approximates the optimal values sufficiently well, optimal behavior can be achieved by following the greedy policy, which at each point in time chooses the action with the highest estimated Advantage value.

Here we are dealing with a partially observable domain. The robot's sensory input is preprocessed using the event extraction mechanism, but different hallways may still look the same and lead to one extracted concept "hallway". This is why the function approximator is a *recurrent* neural network, an LSTM network. The robot's world state is approximated by the current event, provided by the ARAVQ network, together with the recurrent activations within the LSTM network, which can represent a complex function of the entire history of past events. LSTM has proven to outperform traditional recurrent neural networks as well as other short-term memory architectures in learning complex, long-term dependencies on past events [7, 1]. This can be attributed to LSTM's property of non-vanishing error gradients with respect to past events, and its corresponding ability to store information reliably for long periods of time within the so-called memory cells.

2.3 Hierarchical control

The outputs of the reinforcement learning LSTM network represent values of high-level actions in response to changes in events detected by the ARAVQ network. To provide a connection to the low level, real-time motor control of the robot, the whole system is organized as a hierarchical control system. The complete hierarchical control system, inspired by Brooks' subsumption architecture [2], is depicted in figure 3.

Low level, real-time motor control is handled by a simple, handcrafted reactive controller. On this level, there are three behaviors: going forward, following the corridor; following the left wall and turning left when possible; and following the right wall and turning right when possible. Each can correct for some noise in the execution of motor commands and avoid collisions, by directly using the raw sensory inputs as feedback. A low level behavior is executed until a new event is detected by the ARAVQ network. At such a point, the high-level RL-LSTM controller *selects* (or gates) one of the three low level behaviors for execution until the next event.

2.4 Related work

The ARAVQ unsupervised event extraction mechanism [9] was inspired by Tani & Nolfi's event ex-

traction mechanism [15], which in turn was inspired by Schmidhuber's history compression mechanism [12]. Tani & Nolfi's system attempts to predict the next input given the complete history of past inputs, whereas ARAVQ simply attempts to detect changes from the last (stable) input to the next. Tani & Nolfi's system is sensitive to the density of patterns: sometimes a very distinct but rare situation does not get its own concept. Furthermore, they required many iterations of learning. ARAVQ can often learn good concepts in just a single pass through the environment.

The idea of preprocessing raw sensory data for reinforcement learning using an unsupervised learning mechanism has been explored by a number of authors. For instance, Fernández & Borrajo [3] use a standard vector quantization technique to categorize high-dimensional input vectors into a fixed number of concepts, before doing Q-learning on the concepts. They do not perform the type of temporal abstraction employed here; they only perform spatial abstraction over the state space, classifying the current raw sensory input. In fact, such spatial abstraction without temporal abstraction is the focus of a large number of studies in reinforcement learning on generalization [13]. However, in most of that work, unlike our work and that of Fernández & Borrajo [3], there is no separate unsupervised learning component.

Temporal abstraction in reinforcement learning is the focus of another body of literature, which is concerned with hierarchy. In both hierarchical Q-learning [16] and the framework of options [14], high-level actions correspond to entire policies at a lower level, which are executed until a termination criterion in fulfilled. Hierarchical Q-learning is also aimed specifically at POMDPs. Hernandez-Gardiol & Mahadevan [5] make a similar argument to ours, stating that temporal abstraction is particularly important in partially observable reinforcement learning domains in order to deal with long time lags between significant past events and current decisions. Where we use an LSTM recurrent neural network to learn the temporal dependencies between high-level concepts, they use the Nearest Sequence Memory and Utile Suffix Memory algorithms. One important difference is that they use *hand-coded* high-level concepts, whereas we use unsupervised learning to let the robot develop the high-level concepts autonomously.

Finally, there have been a number of studies that have dealt with POMDPs by using recurrent neural networks that learn to approximate value functions [11, 8]. Bakker [1] uses LSTM networks in a similar way as we do here, but in a "flat", non-hierarchical system without event extraction.

3 Experiments

3.1 T-maze

As a first illustration of the ideas presented above, we consider a partially observable T-maze navigation task. Versions of this task have been studied both in a supervised learning context [10, 9] and in a reinforcement learning context [1]. The version investigated here is implemented in a simulation of a Khepera mobile robot equipped with 8 noisy infrared proximity sensors and 2 light sensors, driving around in a continuous environment [9].

The robot's task is to move from the fixed starting position to a goal position (see figure 4). The goal position can either be to the left of the T-junction or to the right. The goal position cannot be perceived, but it depends on a "road sign" that the robot can perceive with its light sensors when it traverses the corridor. Once the robot reaches the goal position, it receives a reward: $r = 4$. If, instead, it reaches the opposite position, it receives a negative reward: $r = -.5$. In both cases, the episode ends and a new one starts, with the goal position assigned randomly to the left or the right. No other rewards are available.

The learning task requires the robot to discover the temporal dependency between the road sign and the action at the T-junction, to learn to store the road sign information reliably in short-term memory, and to learn to use that information at the correct time, while ignoring the irrelevant intermediate sensory inputs—all based on very scarce feedback, a scalar reward when the goal is reached. There are many intermediate sensory inputs, because sensors are sampled at a high rate. And the sensors of the Khepera robot are very noisy, which makes it even more difficult. Thus, despite its conceptual simplicity it is a hard long time lag reinforcement learning problem.

This is a case where the value of event extraction is particularly clear. Figure 4 shows the extracted events during a sample path. Reinforcement learning is done on the basis of the extracted events rather than the raw sensory inputs, using the system described in the previous section. The time lags in the problem are significantly reduced and most of the sensory noise is filtered out. The RL-LSTM network can find the temporal dependency between the road sign and the T-junction relatively easily, because at this higher abstraction level the time lag between them is only two events.

10 runs were performed, using RL-LSTM networks with 4 input units (one for each concept), 3 output units (one for each action), 12 hidden units, and 3 memory cells. The following parameter values were used: $\delta = .7$, $\epsilon = .2$, $n = 5$, $\alpha = .05$, $\gamma = .98$, $\lambda = .8$, $\kappa = .2$, and $\beta = .01$.

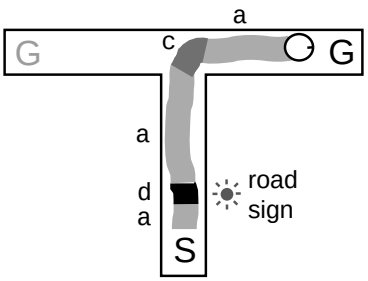

Figure 4: *The T-maze, depicted together with the simulated Khepera robot and the events it detects along the way from the starting position S to the current goal position G (the "wrong" goal position is shown in gray).*

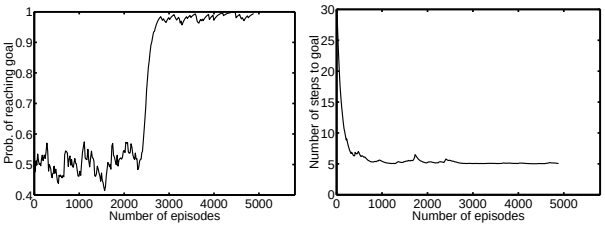

Figure 5: *Left: Probability of reaching the goal in the T-maze task as a function of the number of learning episodes for one run. Right: Average number of high-level actions to the goal.*

All runs converged to optimal policies, using 4081 learning episodes on average. Figure 5 shows the probability of reaching the goal as a function of the number of learning episodes, for one typical run (averaging over all 10 runs is not very insightful because different runs discover and rapidly converge to the correct policy at different moments). It is based on the robot choosing high-level actions according to its exploration mechanism, which is why it does not reach 100% correct; the derived greedy policy is optimal. Figure 5 also shows the average number of high-level actions to the goal, provided the goal is reached, as a function of the number of learning episodes for the same run.

3.2 Complex maze

The second test problem is a more complex maze navigation task (see figure 6). The principle, however, is the same. The agent must move to one of two possible goal positions, depending on a road sign it can only observe near the starting position. Again, $r = 4$ when the robot reaches the goal, and $r = -.5$ when it reaches the opposite position. But now there are many more states and longer paths to the goal, and ARAVQ finds more concepts. Furthermore, even if the robot reliably learns to move from the starting position to the final T-junction before the goal position, the minimum time lag between the road sign and the final T-junction is 6. Thus, although the time lags are considerably reduced thanks to ARAVQ, this is still a long-term dependency POMDP—but now even on the level of extracted events rather than the level of low-level sensory inputs. A similar task was investigated in [9], using a supervised learning Elman recurrent neural network to learn the high-level actions. The Elman network could not solve the task when the time lag exceeded a few events. LSTM, on the other hand, can solve such tasks [7, 1], and it does so in this particular task.

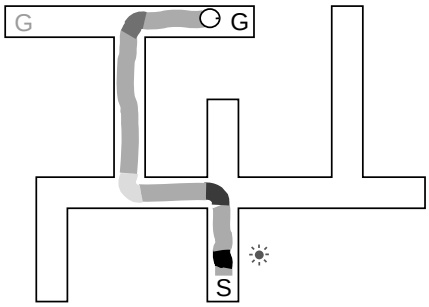

Figure 6: *The more complex maze, depicted together with the robot taking the optimal path to the right goal position. The number of events between the road sign and the final T-junction is 6.*

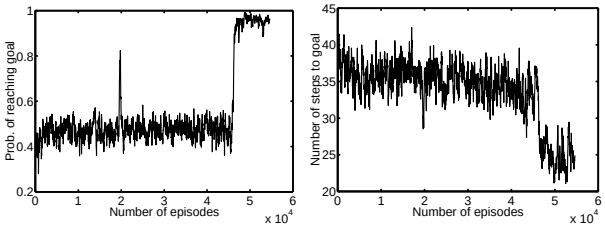

Figure 7: *Left: Probability of reaching the goal in the complex maze task as a function of the number of learning episodes for one run. Right: Average number of high-level actions to the goal.*

10 runs were performed, using the same RL-LSTM architecture as in the simpler T-maze, except the number of input units is now 10, because there are now 10 concepts. All runs converged to optimal greedy policies, always reaching the correct goal position in the lowest possible number of high-level actions. The system needed considerably more time than in the simple T-maze, taking 235,785 learning episodes on average. Figure 7 shows the probability of reaching the correct goal position as a function of the number of learning episodes for one typical run. Figure 7 also shows the average number of high-level actions to the goal, provided the goal is reached, as a function of the number of learning episodes for the same run.

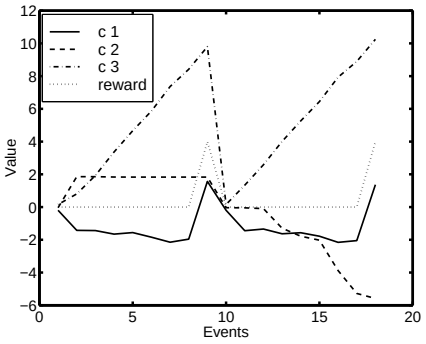

Figure 8: *The memory cells' internal states c_j during two episodes, the first with the goal to the left and the second with the goal to the right. The reward signal is also shown. After the first reward, the new episode starts (event 10).*

For the run depicted in figure 7, figure 8 shows the values of the memory cells' internal states c_j during the episode, *after* the optimal policy has been learned. The robot first encounters an episode where the goal is in the left position, and then an episode where the goal is in the right position. It is apparent that when the robot sees the "left" road sign the internal state of memory cell 2 takes on a different value than when it sees the "right" road sign. This internal state uniquely characterizes the two conditions throughout the episode, and it is used by the network at the final T-junction to decide whether to turn left or right.

4 Discussion

This paper shows that certain learning tasks in continuous, partially observable robot domains which are very difficult in their "raw" versions can be solved using the combination of reinforcement learning with unsupervised learning. The key elements are the ARAVQ event extraction network that accomplishes a spatial and temporal abstraction of the environment, the RL-LSTM recurrent neural network that learns the POMDP's temporal dependencies, and their combination into a hierarchical control system.

However, there are still several limitations. First of all, learning is still fairly slow. This is a problem for most reinforcement learning algorithms and most neural networks. This probably reflects the generality and lack of "bias" or built-in knowledge of the used algorithms as much as it reflects inherent problems in the algorithms per se. But this issue still needs to be addressed if we are to apply these techniques to real robots in real-world tasks.

A second limitation has to do with the fact that here event extraction was independent of reinforcement learning. However, we can easily imagine cases where the unsupervised system fails to distinguish perceptually similar but conceptually distinct situations, or conversely, distinguishes perceptually distinct but conceptually similar situations. Ideally, we want event extraction to make only necessary and sufficient distinctions. This could be achieved by having feedback from the reinforcement learning system to the event extraction system, telling it how "useful" the currently extracted concepts are. Such a system would be another step in the direction of a robot that can learn truly autonomously.

References

[1] B. Bakker. Reinforcement learning with Long Short-Term Memory. In *NIPS 14*. 2002.

[2] R. A. Brooks. Intelligence without reason. In *IJCAI'91*. 1991.

[3] F. Fernández and D. Borrajo. VQQL. Applying vector quantization to reinforcement learning. In *IJCAI'99 Workshop on RoboCup*, 1999.

[4] M. E. Harmon and L. C. Baird. Multi-player residual advantage learning with general function approximation. Technical report, Wright-Patterson Air Force Base, 1996.

[5] N. Hernandez-Gardiol and S. Mahadevan. Hierarchical memory-based reinforcement learning. In *NIPS 13*. 2001.

[6] G. Hinton. Invited talk. In *Workshop "The future and prospects of neural networks" at ICANN'99*, 1999.

[7] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9 (8):1735–1780, 1997.

[8] L.-J. Lin and T. Mitchell. Reinforcement learning with hidden states. In *Proc. of the 2nd Int. Conf. on Simulation of Adaptive Behavior*. MIT Press, 1993.

[9] F. Linåker and H. Jacobsson. Mobile robot learning of delayed response tasks through event extraction: A solution to the road sign problem and beyond. In *IJCAI'2001*, 2001.

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

[11] J. Schmidhuber. Networks adjusting networks. In *Proc. of Distributed Adaptive Neural Information Processing*, St. Augustin, 1990.

[12] J. Schmidhuber. Learning complex, etended sequences using the principle of history compression. *Neural Computation*, 4 (2):234–242, 1992.

[13] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT Press, Cambridge, MA, 1998.

[14] R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[15] J. Tani and S. Nolfi. Learning to perceive the world as articulated. In *SAB'98*. 1998.

[16] M. Wiering and J. Schmidhuber. HQ-learning. *Adaptive Behavior*, 6 (2):219–246, 1997.