Building Internal Maps of a Mobile Robot

Branko Šter and Andrej Dobnikar
University of Ljubljana
Slovenia

1. Introduction

Classical approaches to environment modelling of mobile robots consist of geometrical space reconstruction using measurements from robot sensors. This approach could be error-prone due to noise and inaccuracy of the sensors. However, in order to perform path-planning, a mobile robot still requires some kind of representation or world-model, as pointed out in (Tani, 1996) and (Mataric, 1992).
The alternative approach consists of topological modelling of the environment. It leads to a simpler description of the environment, but still preserving essential information. Provided that a type of low-level behaviour is provided, a topological map of the space suffices to the robot to distinguish among qualitative distinct behaviours. A model of the world may be built therefore on top of the reactive behavior. This corresponds to topological navigation. A topological map of the space is usually in the form of a graph or a finite state machine (FSM). The states are typically distinctive places in the space, also called landmarks. At each node, there is a decision to be made as to where to proceed.
It was argued (Brooks, 1991) that for truly intelligent agents the type of representation must not be a designer's choice, because the abstraction is the key part of the intelligence. Human designers tend to decompose the problem to the blocks-world, i.e., they do all the abstraction and leave a supposedly intelligent agent merely to search in this simplified world. Behavior-based robotics (Brooks, 1991) consequently emerged as a paradigm, emphasizing direct embodiment of a robot in a surrounding environment, taking as the basic level a direct state-action mapping or reactive behavior that enables the robot to perform basic tasks in the real world. Further levels are to be built incrementally upon the basic level, but always having in mind strong coupling between the robot and the environment. In (Mahadevan & Connell, 1992), one of the first successful applications of reinforcement learning to a behavior-based robot is described.
Basic skills of a robot include obstacle avoidance and approaching a goal. While the obstacle avoidance skill is a reactive-type behaviour, i.e., it requires no memory, approaching a goal usually requires some knowledge about the environment. Unless the goal is observed at a given moment, the robot must have a model of the space.
The symbolic approach is simple, but also has a major drawback, namely, input symbols to the finite state machine may occasionally be overlooked or even illegal. Whenever an error occurs (due to sensor noise, for example), this symbolical representation can consequently break down.

(Tani, 1996) applied recurrent neural networks to model the environment. A robot with range sensors advances reactively according to the maximum of the smoothed range profile. When multiple maxima appear, it decides where to proceed (this is a graph node). This approach avoids a serious drawback of the methods that require localization and the position of the robot in global coordinates. The robot does not have to perform localization, since the position is implicitly contained within the current state information in the recurrent neural network (RNN). (Tani, 1996) showed that the robot, whenever it is lost, gets "situated" again, i.e., it eventually restores the correct state information (context). On the other hand, a RNN tolerates a certain amount of errors, thus overcoming the FSM approach.
The approach to handle independently low-level or reactive behavior and higher-level or planning behavior has become common in mobile robotics. The key difference between the two is the usage of memory and different granularity. Reactive behavior (Mahadevan & Connell, 1992, Krose & van Dam, 1992a, Krose & van Dam, 1992b) is responsible for reacting to current sensory information with appropriate action or motor-control. It requires no memory, therefore a purely feed-forward scheme can be applied. The first concern is always to avoid collisions with obstacles and thereby to prevent the damage on the robot. Other goals typically include approaching objects, maximizing the length of the path, etc.
The higher-level or planning behavior (Tani & Nolfi, 1999) may be therefore manipulated either by a FSM or using a model such as RNN, capable of modeling FSMs. The environment as experienced by a moving robot is treated as a dynamical system. Simple types of reactive behavior are supplemented with eventual decisions to switch among them. Switching criteria in fact define states of the FSM. Since it is embedded in the environment and dependent on the sensory flow of the robot, the notion of Embedded flow state machine (EFSM) has been introduced (Ster & Dobnikar, 2006). It is a FSM-like model, embedded in the environment and symbolically representing the sensory flow experienced by the robot when acting according to a certain type of reactive behavior and making decisions in those situations that satisfy certain pre-specified conditions or switching criteria.
An EFSM can be implemented with a RNN which is trained on a sequence of sensory contents and actions and subsequently used for planning. The EFSM is applicable to multi-step prediction of sensory information and the travelled distances between decision points, given a sequence of decisions at decision points. One of the main virtues of this approach is that no explicit localization is required, since the recurrent neural network holds the state implicitly. An important issue regards the ability of this approach to reliably predict the sensors and the traversed distance enough steps ahead.
This chapter is basically divided into two parts. The first considers learning of appropriate reactive or low-level behavior. This does not mean that a low-level behavior cannot be deterministically programmed, however, but learning is potentially more flexible, due to possible changes in the environment. The second part considers the planning behavior by building an implicit model of the environment using RNNs.

2. Learning Reactive Behaviour of a Mobile Robot

We describe an application of the reinforcement learning paradigm to learning reactive behaviour of a simulated mobile robot, equipped with proximity sensors and video color information. The value function is approximated using radial-basis functions, which are adaptively allocated in the input space. The presented approach combines multiple goals in reinforcement learning using modularity.

2.1 Q-learning in continuous space

Q-learning (Watkins & Dayan, 1992) is a variant of reinforcement learning which is appropriate for systems with unknown dynamics. A model of the system is built implicitly during learning. Q-learning is proved to converge only for discrete systems. For continuous systems it usually works by choosing an appropriate approximation architecture, which may be task-dependent. The update equation is

$$\Delta Q(s_t, a_t) = \eta_t \left[g(s_t, a_t, s_{t+1}) + \gamma \min_{a_{t+1}} (s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{1}$$

where s is state, a is action, η is learning step, and γ is discount factor. In our case the state space is continuous, while the action space is discrete and consists of three actions: turn left, forward and turn right. It must be emphasized, that our system is actually not a Markov decision process, since the sensors do not provide the complete state information.
As a function approximator, the radial-basis function (RBF) neural network is used. As a local-basis function, the Gaussian is taken. Its activation on input $x_t \in R^n$ is

$$\Phi_i(x_t) = e^{-\frac{1}{2}\left\| M_i (x_t - c_i) \right\|^2} \tag{2}$$

where n is the joint dimension of the sensory space, $c_i \in R^n$ is the center and M_i is the scaling matrix of the Gaussian. The basis functions are normalized as

$$b_i(x_t) = \frac{\Phi_i}{\sum_j \Phi_j} \tag{3}$$

which causes the approximator to be a kind of look-up table with soft transitions. The Q-values are represented as

$$Q_j(x_t) = \sum_i w_i b_i(x_t), \quad j = 1,2,3. \tag{4}$$

(Samejima & Omori, 1999) proposed adaptive state space construction method, arguing that methods using look-up tables suffer from the curse of dimensionality, while when using global-basis functions the convergence can suffer. Their adaptive basis division algorithm reportedly solved the collision avoidance problem using a smaller number of basis functions.
A similar method called Adaptive basis addition with fixed size basis (ABA-F) (Samejima & Omori, 1999, Anderson, 1993) is applied. It incrementally allocates basis functions in the observation space. A new basis function is allocated when the TD error exceeds a threshold value, $\varepsilon_{TD} > \theta_e$, and there is no basis function near the present location in the observation space, $\phi_i(x_t) < \theta_a$, $i = 1, \ldots, n$. Otherwise, the output weights are updated:

$$\Delta w_{ai} = \eta_t \left[g(x_{t+1}) + \gamma \min_{a'} Q(x_{t+1}, a) - Q(x_t, a) \right] b_i(x_t), \quad i = 1, \ldots, n, \tag{5}$$

where a is the selected action. During learning actions are selected randomly (exploration phase), while later (exploitation phase) the action with the minimal Q value is chosen at any given moment.

2.2 Reactive behavior with modular reinforcement learning

Various modular approaches have been proposed (Karlsson, 1997, Uchibe et al., 1996, Kalmar et al., 1998). In our work, two reactive or low-level modules were defined: the collision-avoidance module (named module C) and the object-approach module (named module A). For each module the activation is defined as a measure of how much a current sensory information has to do with the module. In our case the activation of the module C is 1 everywhere, where proximity sensors (*ps*) are activated, i.e., where obstacles are near the robot, and 0.01 otherwise (in order to prevent the activations of both modules to be zero).
The module A is disactivated everywhere, where the red color (from "color sensors" *cs* – this is not a video camera) is not sufficiently observed, $\sum_{i=1}^{NC} cs_i < 0.1$. Otherwise, act$_A$ is proportional to the above sum. To bring the robot nearer to biological organisms, a function called weariness (*wr*) was defined.
We know that living beings are naturally interested in exploring the surrounding world. They are especially drawn to "interesting" objects or phenomena, i.e., those with higher levels of sensorial activation. The (colored) object approach module reflects this natural curiosity. The reward the robot gets in the vicinity of a colored object, corresponds to the pleasure of a person satisfying its curiosity. The weariness function models eventual diminishing of curiosity, due to the so-called *stimulus satiation*. Ster (2004) describes these phenomena using various theories of motivation from psychology.
The robot, therefore, while wandering around, approaches colored objects, but leaves them after a certain period, since it gets weary. When its weariness reaches a certain plateau, its activation (or motivation) decreases quite rapidly. Consequently, the collision avoidance module overcomes and takes the robot away from the object. Of course, the weariness effect requires memory, i.e., module A cannot remain purely reactive. The weariness is modeled as

$$wr \leftarrow (1-\varepsilon)wr + \varepsilon \sum_{i=1}^{NC} cs_i \tag{6}$$

and the activation is a descending function of *wr*, e.g.

$$act_a = \begin{cases} 1, & wr \le P_1 \\ \dfrac{P_2 - wr}{P_2 - P_1}, & wr > P_1 \end{cases} \tag{7}$$

We chose ε = 0.1, P_1 = 2, and P_2 = 3. These values depend on particular activations. How to define them, is probably not an easy issue.
The two modules are combined in the following manner

$$Q(s,a) = act_C Q_c(s,a) + act_A Q_a(s,a) \tag{8}$$

During exploration each module learns independently its action-value function Q. The activations are not used at this stage. In the exploitation phase greedy actions are chosen w.r.t. Q, the joint Q-values. However, a modular architecture is not meant to be capable in principle of achieving better solutions than a monolithic one. The actual reason for modularity is merely to facilitate learning. Simpler networks escape local minima easier than larger monolithic networks. The drawback is that a modular architecture may yield sub-optimal solutions, when modules are not chosen appropriately.

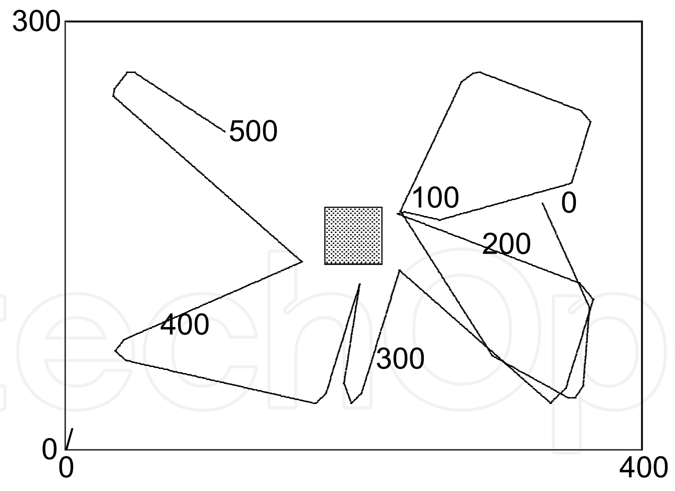

Figure 1. A test path in the environment E1, 500 steps long. The numbers indicate time

2.3 Experiments

The method was tested in two environments, E1 (Fig. 1) and E2 (Fig. 2). The learning (exploration) period lasted 5500 time steps. There was no attempt to minimize this number. In the learning period the robot selected random actions between three available actions: turn left by 0.5 rad (≈ 28.6°), forward by five units, and turn left by 0.5 rad. The actions correspond to the motor commands for the left and the right wheel: left (*dl* = -5, *dr* = 5), forward (*dl* = 5, *dr* = 5), and right (*dl* = 5, *dr* = -5). Since the radius of the robot was 10 units, angle=arc/radius=5/10 = 0.5 rad. The testing period was 500 steps long.

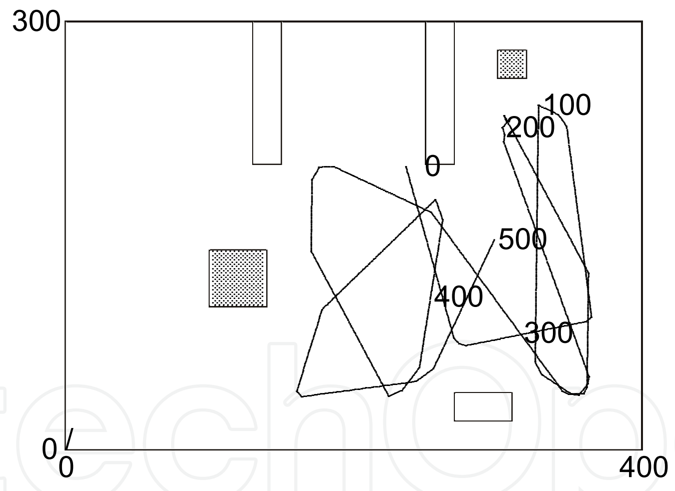

Figure 2. A test path in the environment E2, 500 steps long. Red objects are shaded, others (“pure” obstacles) are empty

The robot is assumed to have eight proximity sensors at angles 90, 45, 0, 0, -45, -90, -180, and -180, similar to the well-known miniature robot Khepera (Mondada et al., 1993). The nonlinear function of the sensors were approximated by a piecewise linear function

$$ps(d)=\begin{cases} 1, & d<10 \\ \frac{50-d}{40}, & 10\le d\le 50 \\ 0, & d>50 \end{cases} \tag{9}$$

where *ps* is the proximity sensor activation and *d* is the distance.

The color information was extracted from 24 range sensors covering a 160° arc in front of the robot, yielding the "intersensorial" angle of about 7°. Each range sensor returns a color, specified in the environment file. In the presented environments a color was either default (grey) or red. This information was compressed into 8 values, each one covering three adjacent range sensors in the following manner: $cs = \frac{25}{3}\sum_{i=1}^{3}\frac{1}{d_i}$.

Parameters of the ABA-F method were: $\sigma = 0.5$, $\theta_e = 0.02$, $\theta_a = 0.1$. The discount factor γ was 0.85. Each collision was penalized by $g = 1$ and each turn action by $g = 0.1$. Seeing the red color was rewarded by a negative penalty $g = -\sum_{i=1}^{8} cs_i \left(1 - \left|i - \langle i \rangle\right| / 4\right)$, where <.> denotes the mean operator. The second term (in brackets) was added in order to reward the side color sensors less than the frontal color sensors.

We show here merely the results of the test run on E2 (for E1 see Ster (2003)), which show Q-values on 200 (out of 500) steps (Qc in Fig. 3, Qa in Fig. 4, and Q in Fig. 5). The actions are labeled as 0 for left, 1 for forward, and 2 for right. It can be seen that the Q-function of the two turn actions is at least 0.1 (mostly about 0.1), because the "turn penalty" was 0.1. The Q-values for forward action, Q(*s*,1), can be lower in an open space, but reach high values in the vicinity of obstacles (note that the colored object is also an obstacle besides being an object of interest), see steps around 90 and 200. At the same points a reward (g < 0) was delivered because of the colored object.

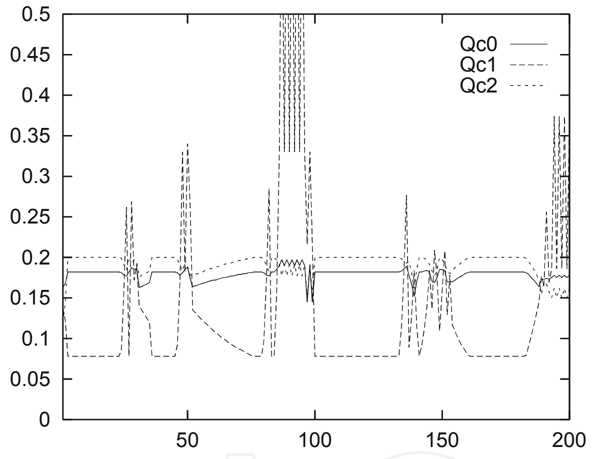

Figure 3. The Q-values of module C over 200 testing steps in E2

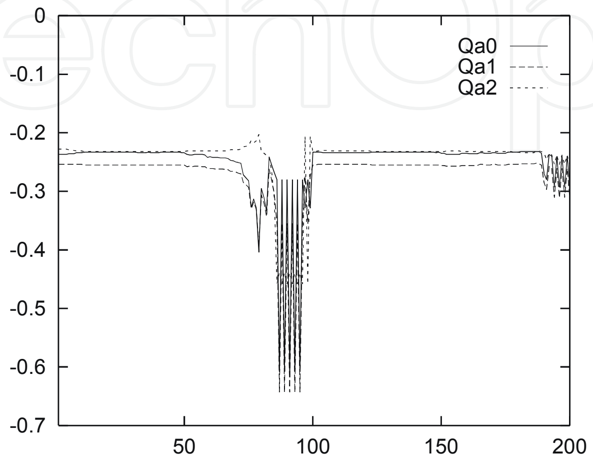

Figure 4. The Q-values of module A over 200 testing steps in E2

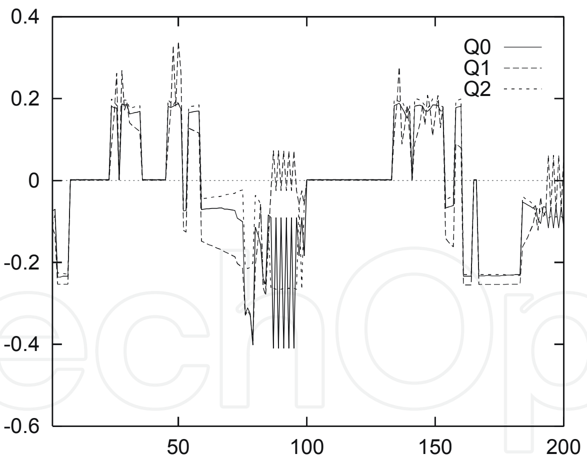

Figure 5. The joint Q-values over 200 testing steps in E2

3. Navigation of a Mobile Robot with Recurrent Neural Networks

When the robot moves according to a specified reactive behavior, a graph node may be implied according to fulfillment of a specified condition, based on sensory contents. This means that a topological graph may be defined implicitly by given switching criteria. In the exploratory or training phase, a random binary action is chosen at such branching point. In our case, one type of action will always be connected with the wall (action 0: follow the wall or return to the wall), and the other will be connected to colored objects in the environment (action 1: approach the object or look for another object). In case of different reactive behavior, of course, actions should be defined appropriately.

The Embedded flow state machine (EFSM) describes the dynamics of this process more accurately. The purpose of the EFSM is to represent a higher-level model of the sensory flow of a moving robot in an environment. The Embedded flow state machine may be formally defined as

$$EFSM = \{S, C, A, D, \delta, \lambda_1, \lambda_2\} \tag{10}$$

where S is a vector space $[0,1]^{Ns}$ of the preprocessed sensory information, C is the context vector space $[0,1]^{Nc}$, A is a finite non-empty set of possible decisions, and D is the unit interval [0,1], representing normalized distance travelled from the previous to the current decision point. N_s is dimensionality of the preprocessed sensory information and N_c is dimensionality of the context space (number of context units, as will be seen later). Functions δ, λ_1 and λ_2 provide the context, the sensors, and the travelled distance at the next decision point. From the context vector $\mathbf{c}_t \in C$, the sensory vector $\mathbf{s}_t \in S$, the distance $d_{t-1,t} \in D$, and the decision $a_t \in A$, all at time t, the next values are predicted:

$$\begin{aligned} \mathbf{c}_{t+1} &= \delta(\mathbf{c}_t, \mathbf{s}_t, d_{t-1,t}, a_t) \\ \mathbf{s}_{t+1} &= \lambda_1(\mathbf{c}_t, \mathbf{s}_t, d_{t-1,t}, a_t) \\ d_{t,t+1} &= \lambda_2(\mathbf{c}_t, \mathbf{s}_t, d_{t-1,t}, a_t) \end{aligned} \tag{11}$$

Of course, the equalities strictly hold only in the case of perfect prediction. The distance is metric information in this otherwise topological approach. It enables optimization of the predicted path in the planning phase. The preprocessed sensory information $\mathbf{s}_t$ stems, in our case, solely from a video camera.

3.1 Recurrent Neural Networks

It was shown that finite-state automata can be represented by recurrent neural networks. There has been a lot of effort to induce FSMs for regular languages with RNNs on the basis of shown examples (Cleeremans et al., 1989, Gabrijel & Dobnikar, 2003).
One of the classic algorithms for training recurrent neural networks (RNN) (Haykin, 1994) is described in (Williams & Zipser, 1989). The algorithm is gradient following and is applicable for fully connected RNNs, i.e., each unit (neuron) has a feedback connection to each unit in the network, see Fig. 6. The outputs of certain units represent outputs of the network, while the other units are called context-units, because they provide information relevant to sequence-processing problems.

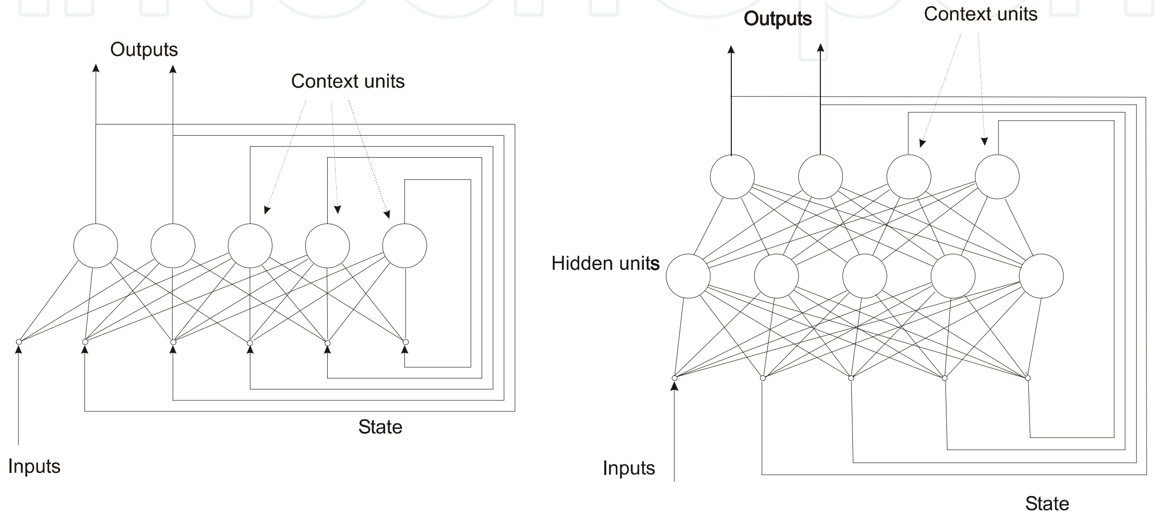

Figure 6. Fully connected recurrent neural network (left) and recurrent neural network with a hidden layer (right)

Inputs to the network are fed to each unit. Thus RNN can be trained to simulate any dynamical system, at least conceptually. The batch variant of the algorithm requires a sequence of input vectors **x**(t) and output vectors **y**(t) for t = 1, ..., *N*, where *N* is the length of the sequence. RNN is trained to produce the sequence of the desired outputs, if fed with the input sequence. A caution must be exercised while training the network; namely, when RNN is trained to produce desired outputs at each cycle, the functional ability of the network is restricted to a single perceptron layer, that is a linear transformation plus sigmoid activation function. To increase the functional ability of the network, inputs and outputs may change after two or more cycles of the network processing. Another possibility is, however, to apply a recurrent neural network with a hidden layer, which increases the functional ability of the network (Fig. 6 right). It is known that the 2-layer perceptron is a universal approximator (Hornik et al., 1989). The modified gradient-based algorithm is described in (Ster & Dobnikar, 2006).

3.2 Building an Internal Map of the Environment

The miniature robot Khepera (Mondada et al., 1993), see Fig. 7, with an additional video color camera was applied. Sensory information consists of infrared proximity sensors, wheel-encoders and processed image from a video camera. The task was considerably simplified using the camera solely for detection of colored objects (red, green and blue in this case). The robot is equipped with eight proximity sensors with a nonlinear activation

function, which can be approximated by a piecewise-linear one, as shown in Fig. 8. The robot is able to perform three distinct actions: turn left, move forward, and turn right (by a small angle).
A video image was thresholded to extract the intensities of red, green and blue colour, which subsequently form the sensory vector. Two images at resolution 80x60, seen by the robot before the thresholding operation, are shown in Fig. 9. They correspond to branching points or EFSM states with the sensory outputs "Green reached" and "Red observed", respectively.

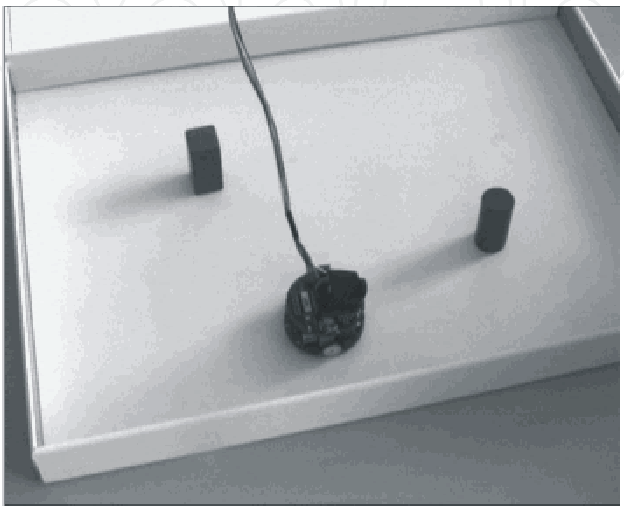
Figure 7. Mobile robot Khepera with an on-board video camera

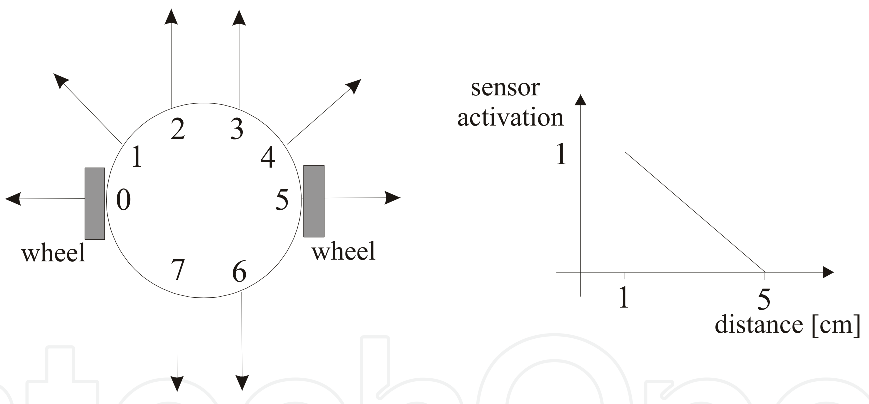

Figure 8. Proximity sensors of mobile robot Khepera

It should be mentioned here that these labels do not actually represent states, but rather the outputs of a finite-state machine, which describes the environmental dynamics. Actual states are really not directly accessible via sensors. For example, the same object may be perceived from different viewpoints and thus the state is different, despite identical or very similar sensorial outputs. Another possibility would be the existence of two identical or very similar objects at different spots in the environment. The corresponding states are different, too.
The low-level or reactive behaviour of the robot consists of following the right wall using information from proximity sensors. The low-level behaviour is deterministic and rather short to program manually, but a little longer to find experimentally. It turned out that the following portion of the low-level program (written in C language) leads to very efficient right-wall following:

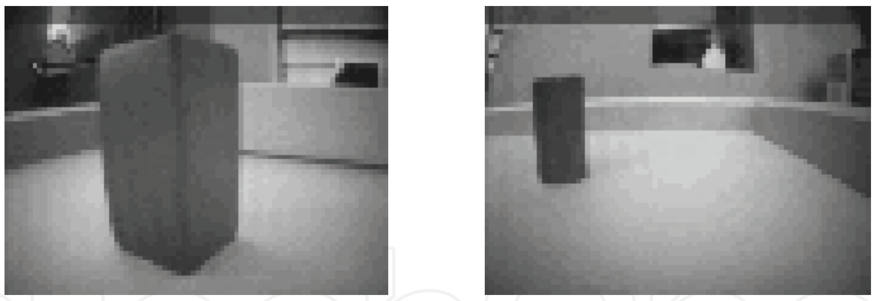

Figure 9. Left: sensory output "Green reached" close to the green object, as seen by the robot. Right: sensory output "Red observed", where the red object is observed, as seen by the robot (resolution 80x60)

```
if ( proxSen[2]>100 || proxSen[3]>100 || proxSen[4]>100 )
   SetSpeed(-h/2,h/2);
else if ( proxSen[5] > 800)        // too close
   SetSpeed(-h,h);                 // left
else if ( proxSen[5] < 300)        // too far away
   SetSpeed(2*h,h);                // right and forward
else
   SetSpeed(2*h,2*h);              // forward
```

The *h* represents the speed and proxSen[0..7] is the vector of proximity sensors. Another possibility is to learn a reactive behavior by applying reinforcement learning, as described in the previous section.

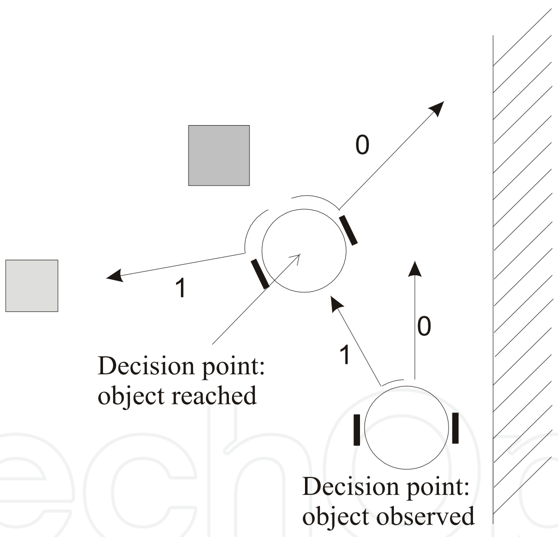

Figure 10. Decision or branching points.

The robot advances following the wall until it observes a colored object. There is a certain, experimentally determined threshold for each colour to signify the observation of an object (about 100 pixels), as well as vicinity of the object (about 600 pixels). When the robot observes a colored object, it has to decide whether to still follow the wall or to approach the object. This situation is called *branching point*. In the vicinity of a colored object, the decision is to approach another colored object or to find the wall (Fig. 10).
Looking for another object, the robot turns left until another object is observed with a specified threshold. It subsequently approaches the object. Looking for the wall, the robot turns right for approximately 90 degrees, and from then on follows Breitenberg's algorithm for low-level

behaviour, which is basically forward-motion, avoiding obstacles. The wheels' velocities are linear functions of the proximity sensors plus an offset term (Breitenberg, 1984).
In this way the whole environment can be represented as an EFSM. EFSMs corresponding to typical indoor environments are usually relatively simple (not many branching points). Therefore, in practical cases RNNs should not have problems learning the corresponding EFSMs. Of course, a complex maze would also produce a complex EFSM.

3.3 Experiments

During training the robot traveled around using random decisions at branching points. The whole path included 101 branching points. An automatic preprocessing, which dealt with situations, where the robot observed the same object again after a short period of time, was done. Thus the path with 87 branching points remained. A small part of the path is shown in Fig. 11. At the first branching point from the start the robot observes object 1 (red). Therefore, the starting sensory output is "Red observed", while the state of the EFSM might be referred to as “1 observed from the west” or O1w (Fig. 12). The training trajectory can be represented as a motor-program describing the action sequence. The vectors, extracted from images, represent the outputs of the EFSM. The outputs do not exactly correspond to the states, so there is not a one-to-one mapping from the states to the outputs. Images at distinct places may look the same, so there is an obvious need to have internal states, i.e., memory.

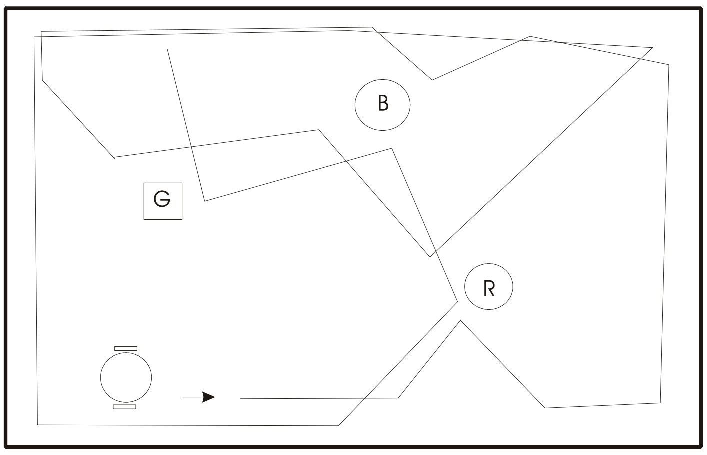

Figure 11. The path of the robot during training

RNN was trained off-line on the sensory-motor sequence for 5000 iterations (RNNs typically require large number of iterations). At each iteration the weights were updated following the gradient of the mean-squared error (MSE) on the whole training set ("batch" variant). The input vector consists of the current sensory vector (red, green and blue intensities), the distance from the previous to the current branching point, and the current decision (action). Given an input vector, a trained RNN predicts the next sensory vector and distance to the next branching point, which can be used in turn as the input at the next branching point, and so on. Given a motor sequence, RNN can thus predict the sensory flow of the EFSM.
We were interested how many steps ahead RNN was able to predict the sensory flow good enough in a closed-loop (multistep prediction). Prediction of the RNN was tested applying all possible motor programs five steps (branching points) ahead, i.e., 2^5 = 32 programs: 00000, 00001, 00010, ..., 11111. Prediction error on the testing set showed that the first four predictions are very accurate, while the fifth is a little less reliable. MSE amounted on

average from 0.01 to 0.02 on the training set and from 0.01 to 0.25 (RMSE from 0.1 to 0.5) on the testing set. However, a few steps correspond to the whole environment, and besides, there was no effort to optimize training of the RNN.

The sense of forward-modeling of the environment is applying it in "mental" planning. The term "navigation" stands for finding the (shortest) path to a specified point. For example, we would like to find the shortest path from beginning to the green object. It is obvious from Fig. 13, that using the program 1-1-1 is the best way to do it. This corresponds to decisions: ‘Approach red’, ‘Find another’ (blue), ‘Find another’ (green). We need to perform a closed-loop simulation of all the motor programs, searching for the matching object, specified as the goal, and finding the least costly path to it. One possibility is proposed also in Tani (1996).

4. Conclusion

The first part of this chapter shows that using reinforcement learning with a modular architecture a mobile robot can learn to wander "sensibly", merely by applying known psychological concepts as sources of reinforcement. It is motivated by distinguishing objects in the environment, unmotivated by weariness, while at the same time avoiding collisions with obstacles and trying to maintain its course. Another possibility is to design reinforcement signals according to specific requirements.

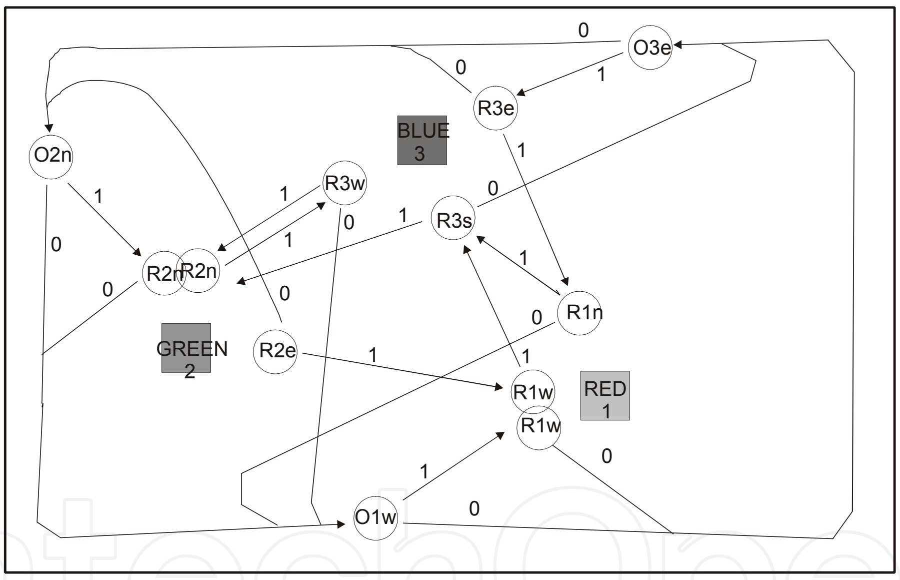

Figure 12. The EFSM, embedded in its environment. There are three landmark objects: 1 (red), 2 (green) and 3 (blue). EFSM states are denoted by circles and labelled as Oi (for 'object i observed') and Ri (for 'object i reached') with additional subscript letters signifying directions, e.g. state R2e signifies 'object 2 reached from the East'

The second part of the chapter shows how recurrent neural networks can be applied to build the internal map of the environment in a simple robot navigation task. The robot is able to induce the EFSM of a simple environment in the form of RNN and to use it for further planning. Because of the simplicity of the environment in the given task, the robot has to predict sensory flow for very short motor-programs, and it does it successfully. Further work should reveal how this behaviour scales up to larger problems and more realistic sensors, i.e., where a lower level of abstraction of sensory information is provided. Also

more reliable and possibly on-line training is required. The ultimate goal would be no human-designed representation, which seems to be still a very complex task.
There is a question whether a RNN is able to induce a finite-state automaton only on the basis of a limited number of training examples. It is obvious that the network can be trained (fitted) to a particular sequence, but it is not clear whether the complete structure of an automaton can be induced. This is not a question of functional ability, since a RNN can simulate any FSM, in principle.
There are two possible types of overtraining or overfitting here. Firstly, due to too many hidden neurons (standard overfitting in static neural networks), and secondly, due to too many context neurons. The latter might be called "dynamic overfitting", namely, the RNN uses its context units to "invent" a kind of context to lower the prediction error on the particular training sequence, especially when the latter is short. This problem would possibly be avoided with an on-line training, long enough to resolve uncertainties. For example, when turning to the left near the red object and looking for another object, the robot may overlook the blue object. The cause would be probably the video camera. In the training sequence there were four occurences of the red object followed by the action "1", i.e., looking for another object (after turning to the left.) Three times the robot observed blue and once green. In the latter case it overlooked blue, unfortunately. In case of larger sensor errors a training sequence should be much longer in order to reliably "collect statistics".

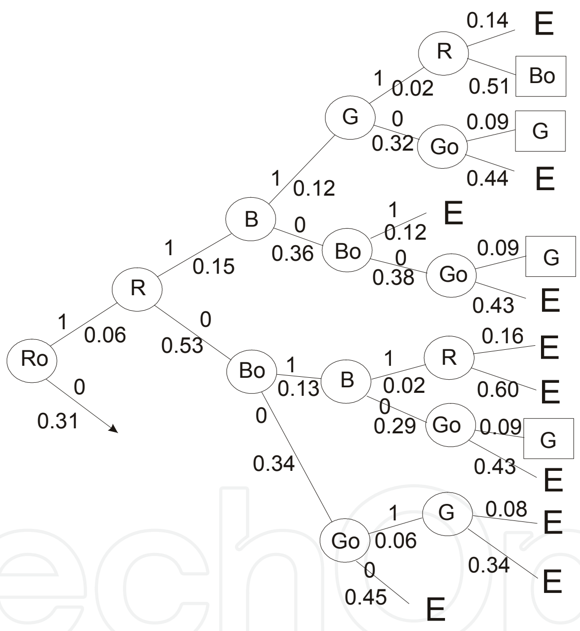

Figure 13. A half of the tree, representing motor programs of the length five. Only the programs, beginning with 1, are shown. At each arc the motor command (0 or 1) and the distance are written. Unclear predictions are labeled with E. Sensory outputs are Ro: red observed, R: red reached, Go: green observed, G: green reached, Bo: blue observed, B: blue reached

5. References

Anderson, C.W. (1993). Q-learning with hidden-unit restarting. Advances in Neural Information Processing Systems 5, San Mateo, CA: Morgan Kaufmann, 81-88.
Arleo, A.; del R. Millan, J. & Floreano, D. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, Vol. 3, No. 1, 9-44.

Breitenberg, V. (1984). *Vehicles: Experiments in Synthetic Psychology*, MIT Press, Cambridge, MA.

Brooks, R. A. (1991). Intelligence without representation. *Artificial Intelligence*, Vol. 47, 139-159.

Cleeremans, A.; Servan-Schreiber, D. & McClelland, J. (1989). Finite State Automata and Simple Recurrent Networks, *Neural Computation*, Vol. 1, No. 3, 372-381.

Gabrijel, I. & Dobnikar, A. (2003). On-line Identification and Reconstruction of Finite Automata with Generalized Recurrent Neural Networks, *Neural Networks*, Vol. 16, No. 1, 101-120.

Haykin, S. (1994). *Neural Networks*, Macmillan College Publishing Company.

Hornik, K.; Stinchcombe, M. & White, H. (1989). Multilayer feedforward networks are universal approximators, *Neural Networks*, Vol. 2, 359-366.

Kalmar, Z.; Szepesvari, C. & Lorincz, A. (1998). Module Based Reinforcement Learning for a Real Robot, In: *Proceedings of the 6th European Workshop on Learning Robots*, Lecture Notes in AI.

Karlsson, J. (1997). *Learning to Solve Multiple Goals*. PhD Thesis. University of Rochester. Rochester, New York, USA. TR 646.

Krose, B.J.A. & van Dam, J.W.M. (1992)a. Learning to avoid collisions: a reinforcement learning paradigm for mobile robot navigation, *Proceedings of the 1992 IFAC/IFIP/IMACS Symposium on Artificial Intelligence in Real-Time control, IFAC*, pp. 295-301, Delft.

Krose, B.J.A. & van Dam, J.W.M. (1992)b. Adaptive state space quantisation for reinforcement learning of collision-free navigation, *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1327-1332. IEEE, Piscataway, NJ, Raleigh, NC.

Mahadevan, S. & Connell, J. (1992). Automatic Programming of Behavior-based Robots using Reinforcement Learning. *Artificial Intelligence,* Vol. 55, 311-365.

Mataric, M. J. (1992). Integration of representation Into Goal-Driven Behavior-Based Robots. *IEEE Transactions on Robotics and Automation*,Vol. 8, No. 3, 304-312.

Mondada, F.; Franzi, E. & Ienne, P. (1993). Mobile robot miniaturisation: a tool for investigation in control algorithms, *Proceedings of the Third International Symposium on Experimental Robotics*, Yoshikawa, T. and Miyazaki, F. (eds.) Kyoto, Japan, October 1993.

Samejima, K. & Omori, T. (1999). Adaptive internal state space construction method for reinforcement learning of a real-world agent. *Neural Networks*, Vol.12.

Ster, B. (2004). An integrated learning approach to environment modelling in mobile robot navigation, *Neurocomputing*, No. 57, 215-238.

Ster, B. & Dobnikar, A. (2006). Modelling the Environment of a Mobile Robot with the Embedded Flow State Machine. *Journal of Intelligent and Robotic Systems*, Vol. 46, No. 2, 181-199.

Tani, J. (1996). Model-Based Learning for Mobile Robot Navigation from the Dynamical Systems Perspective. *IEEE Transactions on Systems, Man, and Cybernetics - Part B:Cybernetics*, Vol. 26, No. 3, 421-436.

Tani, J. & Nolfi, S. (1999). Learning to perceive world as articulated: an approach for hierarchical learning in sensory-motor systems. *Neural Networks*, Vol. 12, 1131-1141.

Uchibe, E.; Asada, M. & Hosoda, K. (1996). Behavior Coordination for a Mobile Robot Using Modular Reinforcement Learning, *Proceedings of the 1996 International Conference on Intelligent Robots and Systems (IROS)*, IEEE Press.

Watkins, J.C.H. & Dayan, P. (1992) Technical Note: Q-learning. *Machine Learning*, Vol. 8, 55-68.

Williams, R. J. & Zipser, D. (1989). A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation*, Vol. 1, 270-280.