CHAPTER 4

LEARNING FINE MOTION IN ROBOTICS: DESIGN AND EXPERIMENTS

C. Versino and **L.M. Gambardella**
IDSIA, Corso Elvezia 36
6900 Lugano, Switzerland
`cristina@idsia.ch`, `luca@idsia.ch`
`http://www.idsia.ch`

Robotics research devotes considerable attention to *path finding*. This is the problem of moving a robot from a starting position to a goal avoiding obstacles. Also, the robot path must be *short* and *smooth*. Traditionally, path finders are either *model-based* or *sensor-based*. While model-based systems address the path finding problem *globally* using a representation of the workspace, sensor-based systems consider it *locally*, and rely only on robot sensors to decide motion. Both methods have limitations, which are rather complementary. By integrating the two methods, their respective drawbacks can be mitigated. Thus, in [15] a model-based system (a planner working on an artificial potential field) and a sensor-based system (a Hierarchical Extended Kohonen Map) which cooperate to solve the path finding problem have been described. Along related lines, several authors [5], [6], [8], [12] have proposed to build automatically the sensor-based system as the result of a learning process, where a local planner plays the role of the teacher. In particular, [5], [8] employ a Self-Organizing Map (SOM) and [6] use a dynamical variant of SOM (DSOM) based on a Growing Neural Gas network [2]. In these works, the decision of using a SOM-like network seems to be justified by its *data topology-conserving* character which is supposed to favor in some way the learning of suitable $< perception, action >$ pairs. None of these works provide experimental evidence for this reasonable, but not obvious, claim.

In this chapter we describe a SOM-like neural network which learns to associate actions to perceptions under the supervision of a planning system. By reporting this experiment the following contributions are made. *First*, the utility of using a hierarchical version of SOM instead of the basic SOM is investigated. *Second*, the effect of cooperative learning due to the interaction of neighboring neurons is explicitly measured. *Third*, the beneficial side-effect which can be obtained by transferring motion knowledge from the planner to the SOM is highlighted.

1 How to Find the Path?

A *path finder* is an algorithm to guide a robot from a starting location to a goal avoiding the obstacles in the workspace (Figure 1).

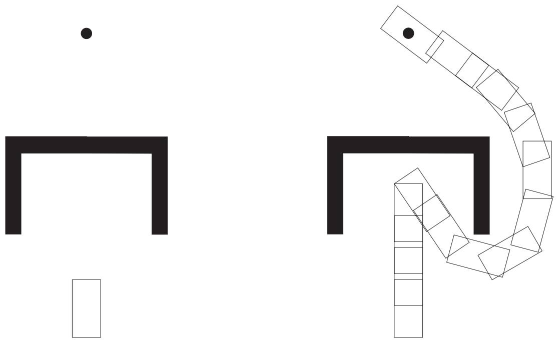

Figure 1. An instance of the path finding problem (left) and a solution to it (right). The white rectangle is the robot, the black circle is the goal, the gray shapes are the obstacles.

A good path finder generates short and smooth paths, and, for this to be possible, it requires both *high level* and *low level* motion skills. At high level, it should be able to reason on the trajectory performed by the robot as a *whole*. This is to recover from dead-ends (Figure 1) and to optimize the path length. At low level, the path finder should be able to decide on each *single step* of the robot trajectory: this is to propose actions that approach the robot to the goal while avoiding collisions. Low level strategies are also referred to as *fine motion* strategies.

Traditionally, path finders are either *model-based* or *sensor-based*. Model-based algorithms use a *model* of the workspace (a map, a camera image or other representations) to generate an obstacle-free path to the goal, while sensor-based algorithms rely only on the robot on-board *sensors* to gather information for motion. Thus, model-based systems address the path finding problem in a *global* way, while sensor-based systems consider it in a *local* way.

Advantages and drawbacks of each method are as follows.

A model-based system is able to compute short paths and to recover from dead-ends, because it works on a complete description of the workspace. However, when such a description is not available, a model-based system is of no use. Also, a model-based system requires considerable computational time to generate the actual robot path, because it needs to evaluate through simulation many alternative paths to select the best one.

A sensor-based system does not need a complete model of the workspace and this is already an advantage in itself. Also, it is computationally inexpensive as it just reacts to sensor readings. But a sensor-based system generates sub-optimal paths, and it may get trapped into dead-ends. Moreover, it is difficult to program a sensor-based system, as we have to predict every possible situation the robot will encounter, and specify a corresponding correct action.

Our research is about the integration of a model-based system and a sensor-based system so that they cooperate and solve the path finding problem in an efficient way [3], [4]. *The focus of this chapter, though, is on the automatic construction of the sensor-based system. Instead of being pre-programmed, this is shaped by a learning process where the model-based system plays the role of teacher. In this way, the programming bottleneck typical of sensor-based systems is bypassed.*

2 The Model-Based System

Traditional model-based systems solve the path finding problem by searching in the space of the robot *free configurations*, the so-called C space [11]. C is made of all robot positions and orientations that are attainable given its kinematics and the obstacles in the workspace. A robot configuration is a multi-dimensional point in C, while the obstacles are forbidden regions in the C space. The drawback of the search techniques based on C is that to determine which configurations are reachable is computationally very expensive.

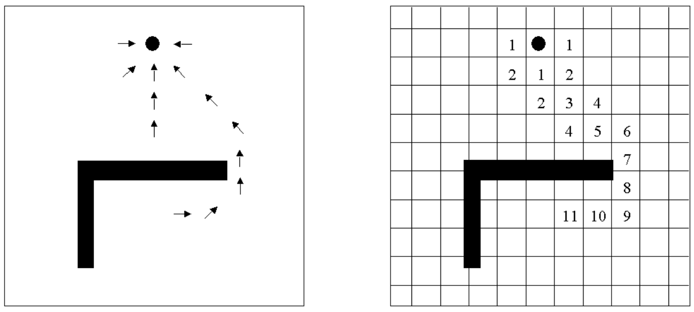

Figure 2. The artificial potential field (left), the field on a discretized workspace.

To reduce this complexity, more recent model-based approaches have suggested to search for a collision-free path by using a "direct" workspace representation of the obstacles and the robot [10]. As an example, a camera image which gives a view from above of the robot in the workspace is a 2-dimensional direct representation (Figure 1, left). Most of the techniques based on direct representations use then an *artificial potential field* V (Figure 2, left) as heuristic to guide the search in the workspace [7]: the robot just needs to follow the field V. At a given location of the workspace, V is the combination of an attractive force towards the goal and of repulsive forces exerted by the obstacles. The computation of the potential field is a one-off step, which needs to be re-executed when either the goal or the obstacles change.

The idea of following the potential field works well if the robot can be modelled as a "point". But when we consider the robot real shape and size we need to extend the potential field metaphor as described below.

First V is created on a discretized workspace (Figure 2, right) and the motion of the robot is guided by a number of *control points* c_i positioned on its shape [10] (Figure 3, left). These points can be thought of as "field sensors". Let $V(c_i)$ be the field value associated to control point c_i, whatever the robot position.

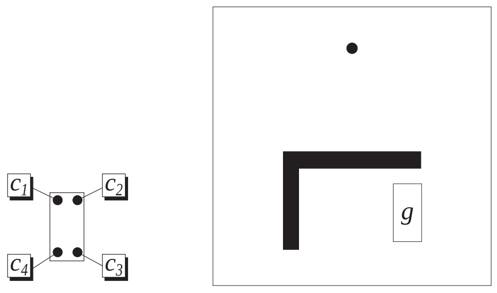

Figure 3. The control points chosen on the robot shape (left). The robot position and orientation in the workspace is its configuration (right).

At any time during motion, the robot *configuration* g is its position and orientation in the workspace (Figure 3, right). When at configuration g, the potential field value V_g of the robot is a combination of the field value of its control points (Equation 1). Roughly speaking, V_g indicates how close is the robot in attaining the goal.

$$V_g = \sum_i a_i V(c_i) \tag{1}$$

Solving the path finding problem is then equivalent to searching for a sequence of collision-free configurations that brings the robot from its initial configuration to the goal. To reduce the search space, the potential field V is used as a heuristic evaluation of the different robot configurations. In what follows, a single configuration transition is described to illustrate how the search process takes place.

Suppose g is the current robot configuration (Figure 3, right). To decide where to move the robot next, i.e., to decide its next configuration g', a set of neighboring configurations is generated by slightly varying g along the robot motion degrees of freedom. This gives rise to, say, a set of four configurations $\{g_1, g_2, g_3, g_4\}$ (Figure 4). Next, for each configuration g_i, its potential field value V_{g_i} is evaluated using Equation 1.

These values are sorted in increasing order: $\{V_{g_1}, V_{g_2}, V_{g_4}, V_{g_3}\}$. The final step is to determine which is the configuration with the smallest potential

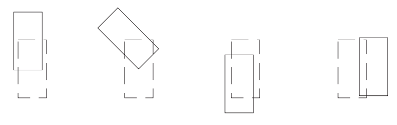

Figure 4. The neighboring configurations.

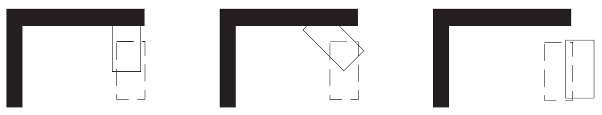

Figure 5. Collision check for robot configurations.

field value that is also collision-free. Since V_{g_1} is the smallest value, we start by simulating g_1 to check whether it produces a collision with some obstacle. Note that this verification is feasible because an image of the robot environment is available, making it possible to explore "mentally" the consequences of an action. After inspection, g_1 and g_2 are discarded (they both collide). Finally, g_4 is accepted as the next configuration for the robot: $g' = g_4$.

It may happen that all neighboring configurations of g produce collisions. This means that the robot is in a *local minimum* of V_g. Typically, a local minimum arises in a dead-end on the robot path (Figure 1). To recover from a local minimum, alternative configurations to the current one can be generated according to various rules, ranging from simply backtracking one step in the robot configuration search-tree to more sophisticated high-level motion strategies [3], [4].

We are now in a position to understand why the planner is computationally expensive. First, it calculates a great number of candidate configurations, because it cannot guess which configurations are acceptable. Second, the order of evaluation of the candidate configurations depends only on their potential values; the robot shape is not taken into account. *The planner does not learn from its own experience*.

In short, a planner is a very general method to solve path finding problems: it may be applied to robots of *any* size and shape. However, this generality is paid in terms of efficiency. When it is foreseen to work with a robot of a *given* size and shape, the flexibility provided by a planning system is less important, and the time needed to plan the robot motion is the main issue. This is why we propose to construct a *custom* sensor-based system, a system which is tailored to a robot of a given size and shape.

3 The Sensor-Based System

Suppose that a robot of a given size and shape is chosen. While using the planner to solve instances of the path finding problem, knowledge for fine motion can be collected and expressed as a set of pairs $< perception, action >$. Each pair refers to a single step of the robot trajectory, and describes a sensory perception (Figure 6) and the action selected by the planner for that perception (Figure 7). To be more specific, the perception is made of a vector o of readings of 24 obstacle proximity sensors, together with the relative goal direction g, a two-dimensional vector of unitary length. The planner action a is a triple representing a x-translation, a y-translation, and a rotation with respect to the robot's current position and orientation. Both the xy-translations and the rotation take discrete values, and can be either positive, negative or null.

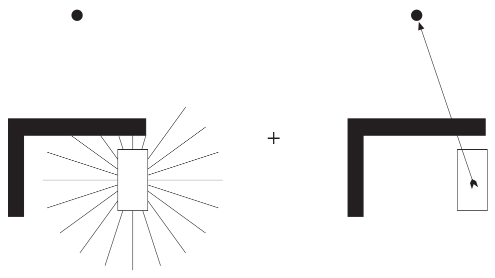

Figure 6. The robot perception: distance from the obstacles (left), relative goal direction (right).

The pairs $< perception, action >$ are the basis to build "automatically' the custom sensor-based system: for this purpose, a *supervised learning*

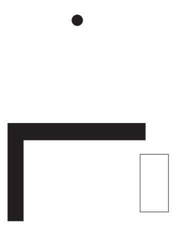

Figure 7. The action selected by the planner.

approach is used. A pair $< perception, action >$ is learnt incrementally by the following steps:

1. the sensor-based system gets the perception as input;
2. it generates a tentative action according to its current knowledge;
3. this action is compared to the target action selected by the planner, and the difference between the two is used to change and improve the behavior of the sensor-based system.

This sequence of steps is repeated for every example available. As a sensor-based system we use an *Artificial Neural Network* (ANN).

Before describing the ANN structure, let us observe that, in general, learning is *hard*. It cannot be achieved just by forcing through the ANN thousands of examples. To facilitate learning, we can take advantage of a good characteristic of the path finding problem, namely that *similar perceptions require similar actions*. This is not always true, but it is true most of the time. As an example, consider the two robot perceptions shown in Figure 8. The only difference between the two is a small variation in the goal position. The action "move to the right" which is suitable for the first case may also be applied to the second.

This property suggests a modular architecture for the ANN, by which its operation is logically divided into two steps (Figure 9). Given an input perception, the ANN *first* determines which is the most similar perception out of the ones experienced so far (step (A)). *Second*, it triggers the action associated to the prototypical perception selected at the first step

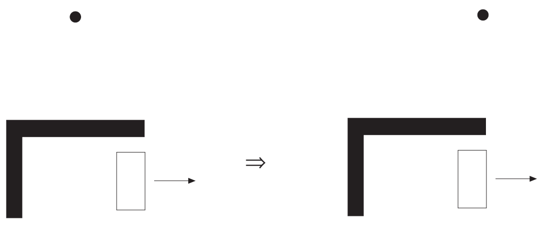

Figure 8. Similar perceptions require similar actions.

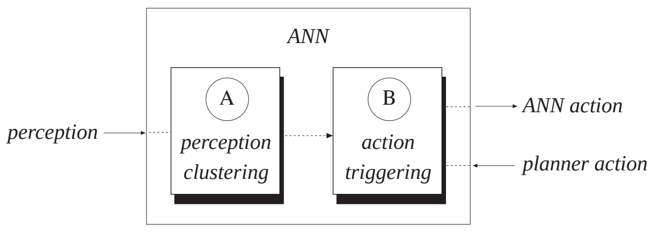

Figure 9. The two step operation of the ANN.

(step (B)). (A) is a clustering operation that aims at grouping together similar experiences into a single “prototypical perception”. In this way, similar perceptions all contribute to the learning of the same (or similar) action, while different perceptions (perceptions which may require different actions) do not interfere one with the other because they are mapped to different prototypical perceptions by the clustering step.

Both step (A) and (B) require learning, and can be globally addressed by the Hierarchical Extended Kohonen Map algorithm. We start by describing step (A).

4 Perception Clustering

In (A) (Figure 9) the ANN learns the concept of similar perceptions. The task is to construct a set of prototypical perceptions out of a set of perceptions experienced by the robot during motion. This clustering task is solved by the basic *Kohonen Map* (KM) algorithm [9].

Kohonen Map. A KM is a two-layered network (Figure 10) consisting of an input layer of neurons directly and fully connected to an output layer. Typically, the output layer is organized as a two-dimensional grid G. w_r is the fan-in weight vector (reference vector) associated to the neuron placed at position r on G.

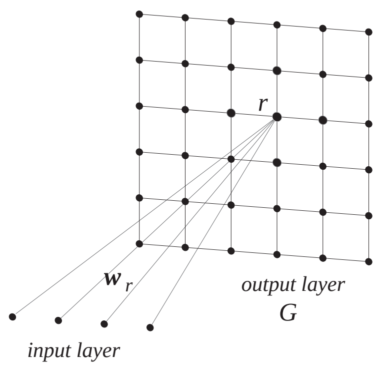

Figure 10. The KM network architecture.

The network is trained by *unsupervised* learning on a set of examples $\{x(1), \ldots, x(T)\}$. For each example x, the following sequence of steps is executed.

1. x is presented to the input layer.

2. A *competition* between the neurons takes place. Each neuron calculates the distance between its reference vector w_r and input pattern x.

$$d(x, w_r) = \|x - w_r\|^2 \tag{2}$$

   The neuron s whose weight vector is the closest to x is the *winner* of the competition.

$$s = \arg\min_r d(x, w_r) \tag{3}$$

3. s is awarded the right to learn the input pattern, i.e. to move closer to it in data space:

$$w_s^{new} = w_s^{old} + \alpha(t) \cdot \left(x - w_s^{old}\right) \tag{4}$$

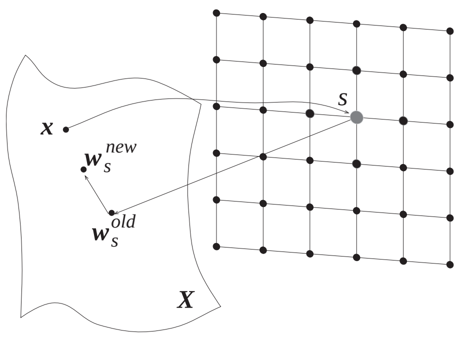

Figure 11. The KM learning step.

Figure 11 illustrates the weight change process of neuron s in the original data space.

In Equation 4, $\alpha(t)$ is the *learning rate*, a real parameter that decreases linearly with the pattern presentation number t.

$$\alpha(t) = \alpha(0) \cdot \left(1 - \frac{t}{T}\right) \tag{5}$$

4. A special trait of the Kohonen algorithm is that the learning step is extended also to the *neighbors* of the winner s. The neighbors of s are those output elements whose distance to s, measured on the grid G, is not greater than a neighborhood parameter $n(t)$. Likewise to α, n decreases linearly with time.

$$n(t) = \left\lceil n(0) \cdot \left(1 - \frac{t}{T}\right) \right\rceil \tag{6}$$

   At the beginning of the learning process, the neighborhood parameter is large, and many neurons share the learning step. As time progresses, fewer neurons are allowed to become closer to the presented input pattern. Figure 12 shows how the neighborhood shrinks in time.

The KM can be used for *data quantization*: an input pattern x can be represented by the reference vector w_s that wins when x is presented to the KM. Also, *similar input patterns are mapped onto neighboring*

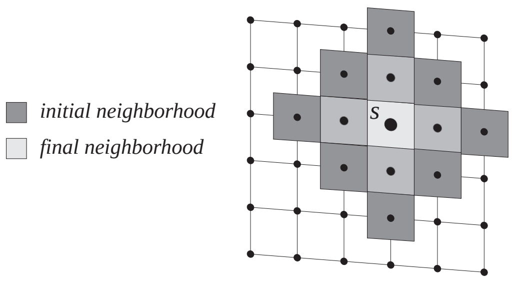

Figure 12. The neighborhood size shrinks as learning proceeds.

locations on the grid (Figure 13). This is because neurons that are close on G are also close in the original data space, a property induced by step 4 of the KM algorithm. Thus, the quantization process preserves the *topology information* contained in the original data. We will show how this topology preserving representation proactively helps the learning of fine motion.

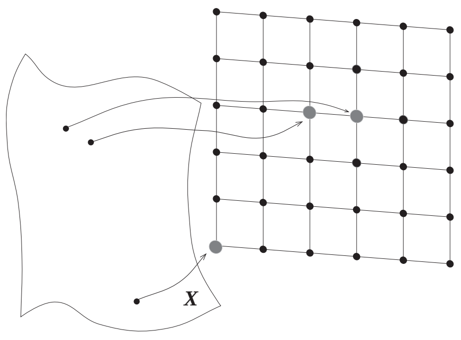

Figure 13. The KM maps similar input data to neighboring locations on the grid.

Back to perception clustering. If a KM is trained with perceptions as inputs, then its reference vectors will represent prototypical perceptions.

To be more specific about the KM structure, remember that a perception is made of a vector o of readings of 24 obstacle proximity sensors, together with the relative goal direction g, a two-dimensional vector (Figure 6). These two vectors could be joined into a single input vector for the KM. If we do so, each neuron in the network will represent an obstacle perception and a given goal direction. However, this representation is not “economic”, as it will be made clear by the following example.

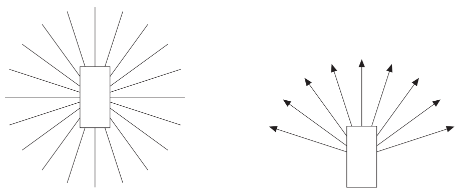
Figure 14. The robot perception while moving in the free space: obstacle perception (left), goal perceptions (right).

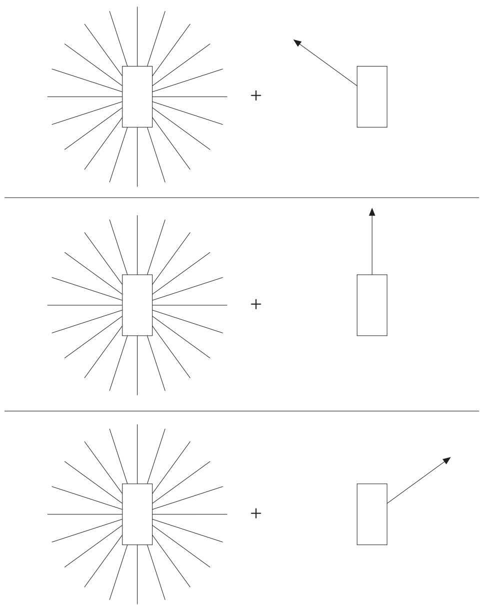

Figure 15. The reference vectors of the trained 1×3 KM.

Imagine the robot is moving in the free space. The only obstacle perception it can experience is that shown in Figure 14, left. In this case, the only varying input parameter is the goal direction. Suppose that the robot has experienced the goal directions shown in Figure 14, right. If we use, say, a 1×3 KM to cluster this input information, the training of the KM would produce the prototypical perceptions depicted in Figure 15. From this representation it is clear that the information concerning the obstacle perception is unnecessarily repeated three times.

To avoid repetitions a Hierarchical Kohonen Map (HKM) [13] may be used instead of the “flat” KM.

Hierarchical Kohonen Map. A HKM is a KM network (*super-network*) composed by a hierarchical arrangement of many subordinated KMs (*sub-networks*) (Figure 16).

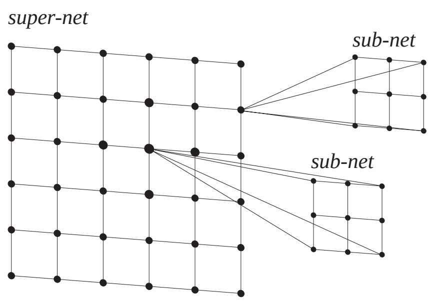

Figure 16. Architecture of the Hierarchical Kohonen Map.

A HKM is used to process input components in *sequence*.

As an example, suppose we want to process an input vector x which is the combination of two component vectors x_1 and x_2. To process x_1 and x_2 separately, we split the input presentation into two steps (Figure 17).

1. x_1 is presented as input to the super-net G. Let neuron s be the winner of the competition on x_1 in G.

2. Then, x_2 is presented as input pattern to sub-net G_s, i.e. the KM associated to the winner s in G. Now, a neuron v in G_s is selected as the winner of the competition on x_2.

The learning rule for the HKM is the same as the one presented for the simple KM. The only difference is that now we have two neighborhood parameters, one for the super-network and one for the sub-nets.

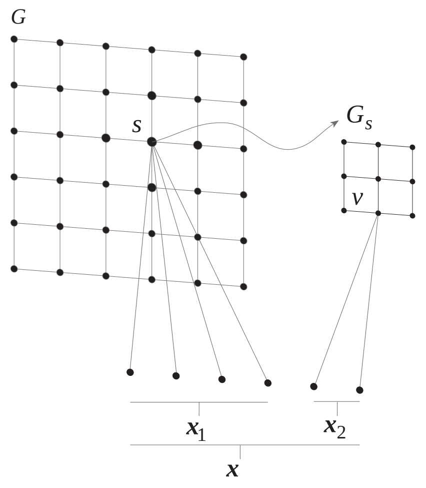

Figure 17. The HKM processes input components in sequence.

In short, in our learning case there are three reasons for preferring a hierarchical architecture to a “flat”. First, it avoids unnecessary repetition of o weights for different g directions, which would be costly in terms of memory requirements. Second, it deals naturally with the economic input representation of g as a 2 dimensional vector. A flat network would need either a more distributed coding for g (as in [12]) or a weighting of g (as in [5], [6]) so that during the matching step g does not lose importance with respect to o, whose dimensionality is rather high. Third, by processing the input information in two stages, we hope to simplify the adaptation process of the SOM to the perception data distribution.

Experimental results on perception clustering. To experiment with the HKM, we designed for the rectangle-shaped robot the workspace shown in Figure 18. In this workspace, the planner solved a number of instances of the path finding problem. Each new instance was obtained by changing the robot initial position while keeping fixed the goal position. In all, the planner generated about 500 pairs $< perception, action >$.

The HKM super-network is a 4×6 grid of neurons, while each sub-net is an array of 10 neurons (Figure 19). For the sub-nets an array arrangement is preferred, because the data to cluster (the goal directions) are points distributed on a circle.

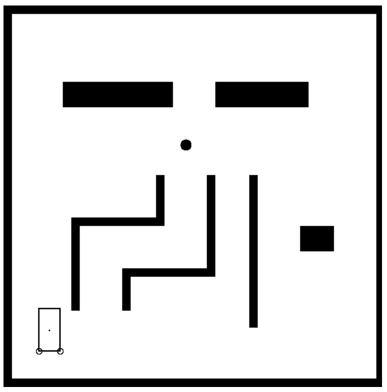

Figure 18. The workspace for the experiments.

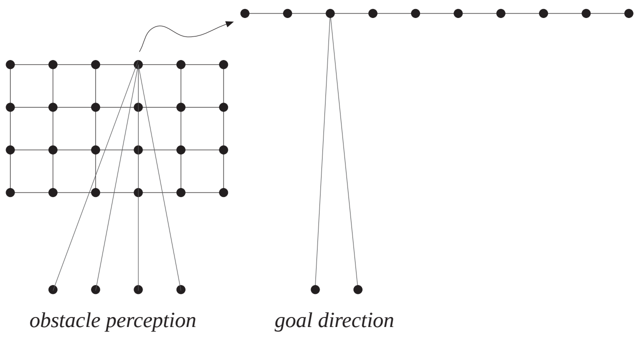

Figure 19. The HKM net structure.

The result of the training phase is depicted graphically in Figures 20–21.

Figure 20 shows the reference vectors of the super-network. They represent prototypical obstacle perceptions experienced by the robot during motion. For example, neuron[1] #0 is the perception of free space, neuron #5 is the perception of a wall on the right-hand side, neuron #7 is the perception of a wall behind the robot's back, and neuron #17 is the perception of a narrow corridor. Observe the *topology preserving* character of the KM: the perception similarity varies in a continuous way on the map. Therefore, similar perceptions activate neighboring locations on the grid.

[1]Neurons are numbered from the upper-left corner to the right.

Figure 21 shows the reference vectors of three sub-nets, namely those corresponding to neurons #0, #7, #17 in the super-network. They represent prototypical goal directions experienced by the robot for the corresponding obstacle perception. The goal direction has been represented as a white vector departing from the center of the robot shape. Again, the *topology preserving* character of the network can be appreciated: the goal direction varies on the array structure in a continuous way.

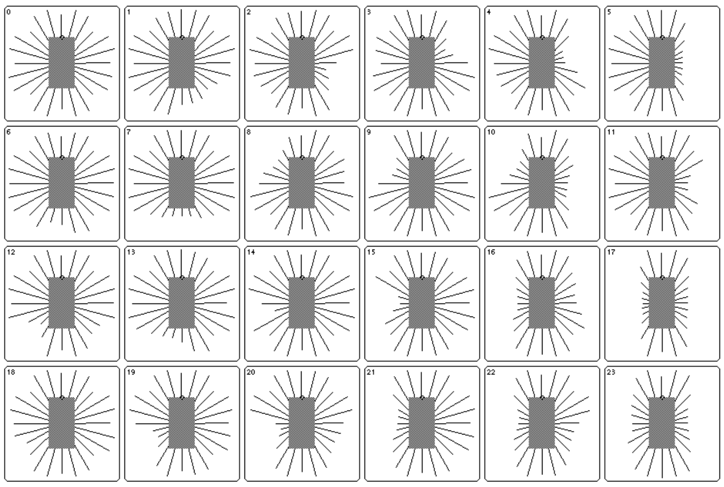
Figure 20. The obstacle perceptions learned by the super-network. (From [15]. With permission.)

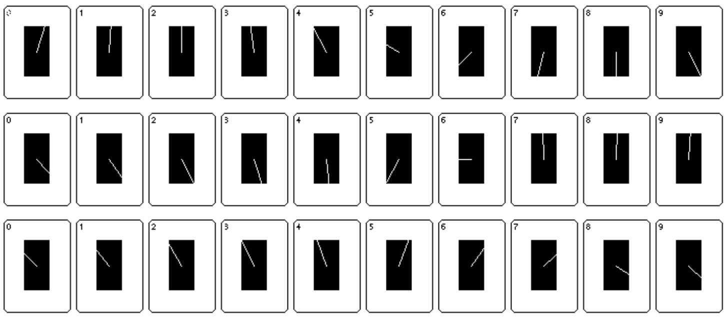
Figure 21. The goal directions learned by three sub-nets. (From [15]. With permission.)

5 Action Triggering

In step (B) (Figure 9) the ANN learns to associate the planner actions to prototypical perceptions. This learning phase takes advantage of the ordered perception representation generated by the HKM at previous step. We introduce briefly the required extension to the basic KM algorithm (EKM) [14], which makes it possible to train the KM network by supervised learning.

Extended Kohonen Map. From the architecture point of view, the KM network is augmented by adding to each neuron r on the competitive grid G a fan-out weight vector z_r to store the neuron output value (Figure 22).

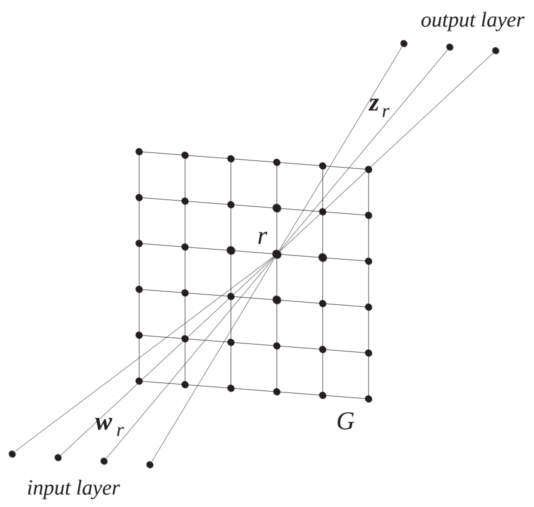

Figure 22. Extended Kohonen Map architecture.

The computation in the EKM network proceeds as follows. When an input pattern x is presented to the input layer, the neurons on G compete to respond to it. The competition involves the neurons fan-in weight vectors w_r, and consists of the computation of the distance between x and each w_r. The neuron s, whose fan-in vector w_s is the closest to x, is the winner of the competition, and its fan-out vector z_s is taken as the network output answer to x.

During the training phase, both the input pattern x and the desired output value y proposed by the teacher are learnt by the winning neuron and by its neighbors on the grid. The learning step consists of moving the fan-in weight vectors of the selected neurons closer to x, and their fan-out weight vectors closer to y (Figure 23, right).

This learning style has been described as a *competitive-cooperative* training rule [13]. It is *competitive* because the neurons compete through their fan-in weight vectors to respond to the presented input pattern. As a consequence, only that part of the network which is relevant to deal with the current input data undergoes the learning process. Moreover, neighboring locations on the grid will correspond to fan-in weight vectors that are close to each other in input data space. The rule is also *cooperative* in that the output value learnt by the winning neuron is partially associated to the fan-out weight vectors of its neighbors. If the input-output function to be learnt is a continuous mapping, then spreading the effect of the learning of an output value to the neighborhood of the winner represents a form of generalization which accelerates the course of learning [13].

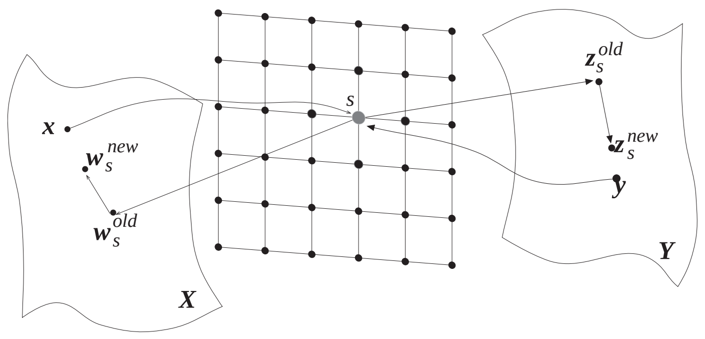

Figure 23. The learning step for the Extended Kohonen Map.

Back to action triggering. To apply the EKM to the action learning problem, we add to every neuron in each sub-network of the HKM a fan-out weight vector that will specify the action to perform for a given obstacle perception and a given goal direction. The complete ANN architecture is a Hierarchical Extended Kohonen Map (HEKM) and is shown in Figure 24.

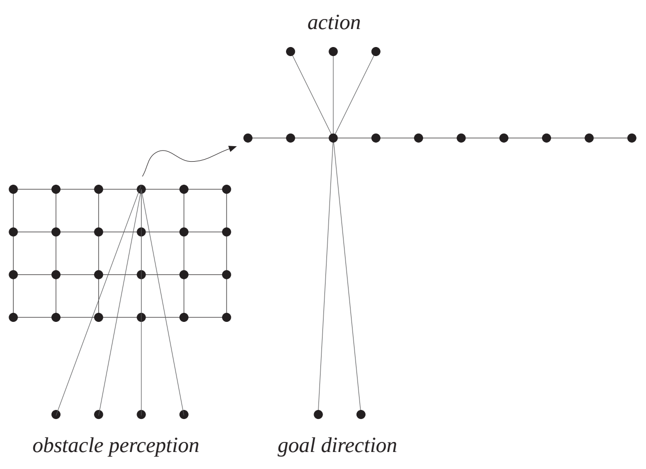

Figure 24. The HEKM network architecture.

Experimental results on action triggering. Some results for the perception clustering module have already been presented. Let us show the corresponding actions learnt by the ANN.

In Figure 25 we have represented the goal directions and the robot actions learnt by sub-network #0, the network responsible for moving the robot in the free-space. In each cell, the gray rectangle is the robot initial configuration, while the black rectangle represents the new robot configuration reached by performing the corresponding action. Similarly, Figures 26–27 show the actions learnt by the sub-networks which are responsible, respectively, to deal with the perception of a wall behind the robot's back, and that of a narrow corridor.

Again the topology preserving character of the network may be appreciated by observing how the learnt actions vary in a continuous way on each sub-network.

6 All Together

Some instances of the path finding problem are presented in Figure 28 together with the solutions found by the ANN in cooperation with the planner. In these trajectories the planner takes control only when the action proposed by the ANN would lead to a collision.

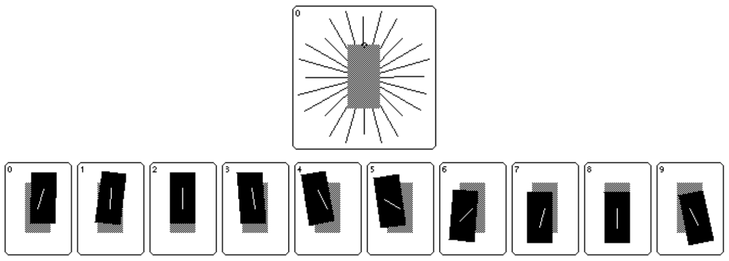

Figure 25. Obstacle perception, goal directions and actions learnt by sub-net #0.

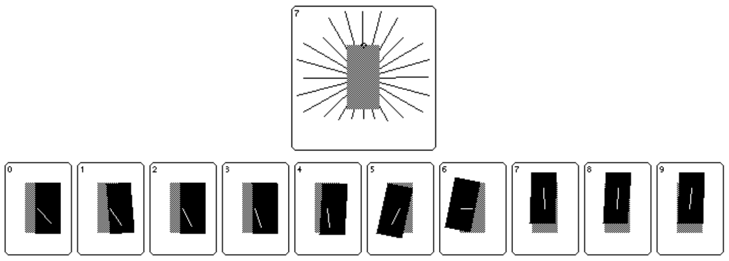

Figure 26. Obstacle perception, goal directions and actions learnt by sub-net #7.

These paths illustrate the ANN ability to deal with several motion situations, namely avoiding a wall to the right-hand side, going around a column, entering doors, going zig-zag.

It is important to stress that, although the ANN has been trained on fine motion with respect to a *fixed* goal, the knowledge it has acquired is "general" because the goal position is not specified in absolute coordinates, but relatively to the robot. The second row of Figure 28 shows the very same ANN guiding the robot to new goal positions.

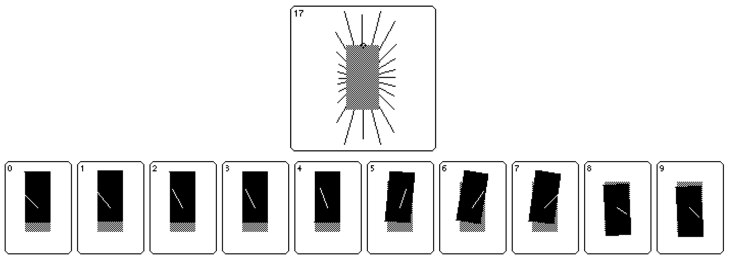

Figure 27. Obstacle perception, goal directions and actions learnt by sub-net #17.

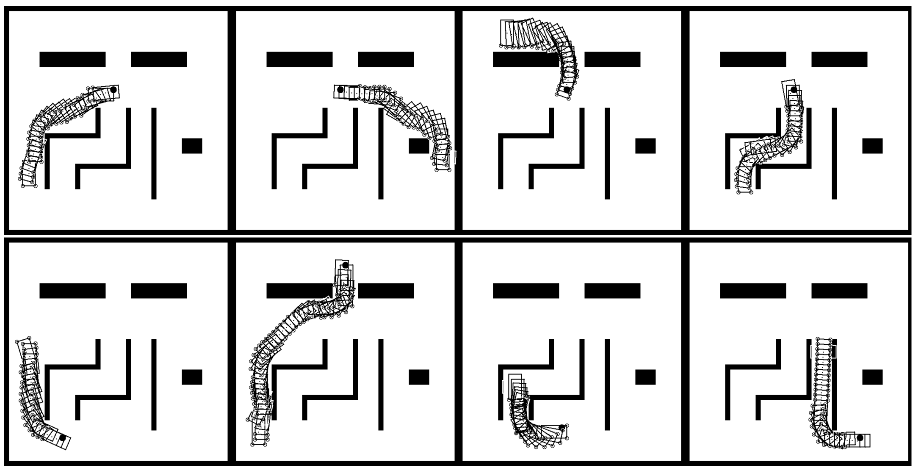

Figure 28. The robot solving the path finding problem with the fixed goal used during the training phase (first row) and with new goal positions (second row). (From [15]. With permission.)

7 Why Use a SOM-Like Network?

We would like now to discuss the following claim: the data topology-preserving character of the HEKM favors the learning of fine motion.

This statement can be proved experimentally by performing two separate training sessions. In the first session, the neighborhood parameters (one for the super-net, one for the sub-nets) are set to 0, while in the second session they are set to values other than 0 (4 and 5, respectively). In this way, the effect of cooperation during learning can be studied.

To evaluate the two methods, an error criterion and a performance criterion are used. The error measure is the mean squared error between the network output action and the target action proposed by the planner, while the performance criterion is the percentage of optimal actions learnt by the network. By definition, the optimal actions are those proposed by the planner.

Let us comment on the plots of error and performance as a function of the number of training cycles (Figure 29). As far as the error is concerned (top plot), one can see that without cooperation (curve with black dots) a certain error level is reached quite rapidly, but afterwards, no significant improvement is observed. On the contrary, with cooperation (curve with white dots) it takes more time to reach the very same error level, but the final error is lower. This type of behavior seems to be typical for cooperating agents, as it is reported in [1]. In our experiment, a possible explanation for this could be that, when the cooperation between the neurons is active, it takes more time to find a good “compromise” to satisfy competing learning needs. However, once the compromise is met, the final result gets improved. A corresponding behavior is observed in the performance curves (bottom plot). Without cooperation a certain performance level is achieved quite rapidly (42%), but after that point no further improvement occurs. With cooperation, the same performance level is obtained later, but the final result is more satisfactory (65%).

8 Planner Vs. HEKM

We conclude by highlighting an interesting side effect which is obtained by transferring motion knowledge from the planner to the HEKM.

Our planner is a *discrete* system. By the term “discrete” we refer to the fact that, at each step of the robot trajectory, the planner generates a finite number of neighboring configurations, and chooses among these the one which approaches the goal closest while avoiding collisions. The HEKM, on the contrary, tends to produce actions which look like being *continuous*. That is because the action learnt by the network for a given perception is a kind of average action performed by the planner in similar perceptual states. To illustrate this point, we let the planner and the HEKM solve the same path finding problem as *stand-alone* systems (Figure 30). One can immediately appreciate qualitative differences in the two paths. The discrete nature of the planner is evident in the left plot: the robot motion is optimal in terms of path length, but quite abrupt. On the contrary, in the HEKM path (right plot) smoothness has been traded against optimality. This observation also accounts for the “sub-optimal” performance level reached by the HEKM (Figure 29) at the end of training.

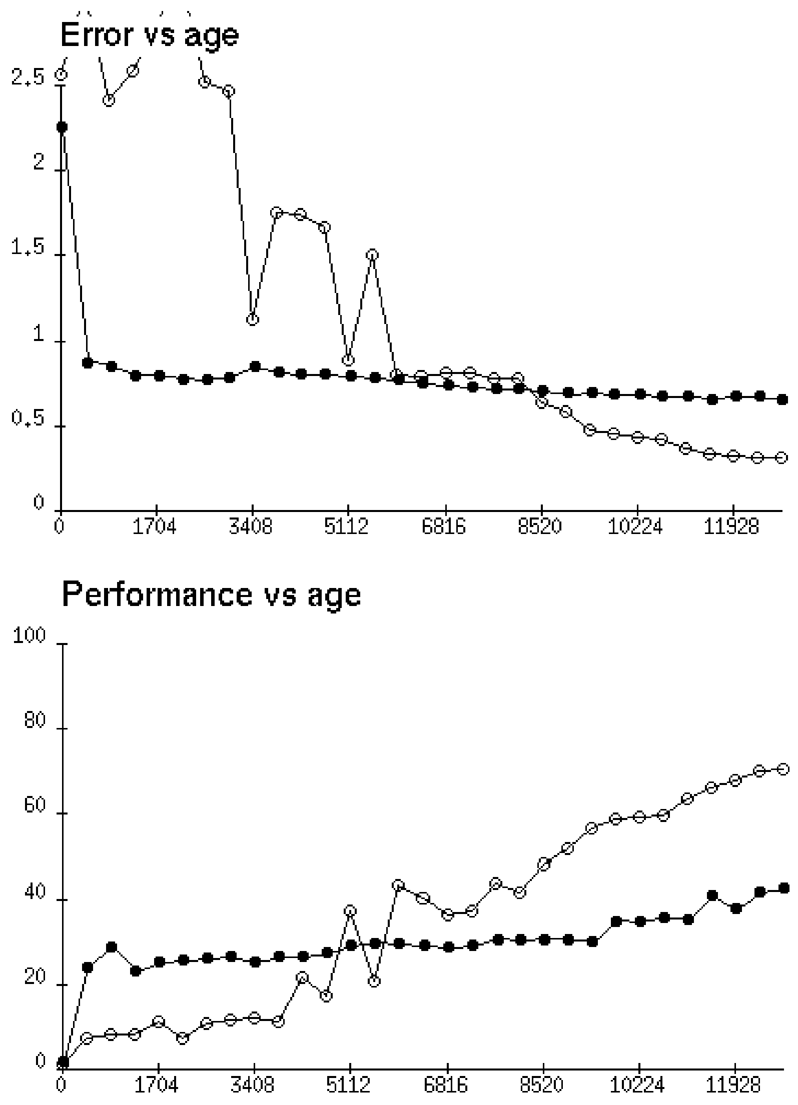

Figure 29. Error (top) and performance (bottom) without cooperation (black dots) and with cooperation (white dots). (From [15]. With permission.)

9 Conclusions

This chapter has presented a HEKM which learns fine motion under the control of a planner.

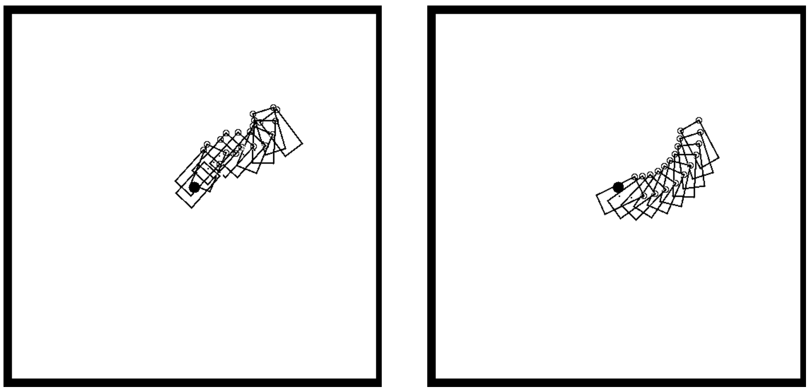

Figure 30. The planner (left plot) and the ANN (right plot) solving the same instance of path finding as stand-alone systems. (From [15]. With permission.)

When invoked, the planner proposes an action, but to do so it needs to explore all possible robot configurations and this is expensive. This is why we capture the planner experience under the form of <*perception*, *action*> pairs and use these as training examples to build automatically the sensor-based HEKM. The result is that the HEKM learns to avoid collisions, acquiring an "implicit" model of the real robot through the perception-action mapping it implements. Of course the HEKM is not fault-free: when the action computed by the HEKM would lead to a collision, control is taken over by the planner, which proposes its action, and this can be treated as a new training example for the HEKM. Therefore, when the HEKM is started as a "tabula rasa", the planner is in charge of proposing the robot action most of the times. But, as the HEKM becomes more competent, control is gradually shifted to the HEKM, and the planner is invoked only in faulty cases. Overall, the integration of the planner and the HEKM improves the performance of the robot in cluttered environments because it decreases both the number of collision checks required and the number of times the planner is activated.

It is also worth noting that the integrated planner-HEKM system is able to take advantage from whatever knowledge is available about the environment. When a model of the workspace is available, we can let the planner be in charge of high-level navigation while the HEKM takes care of fine motion. But if the environment is unknown, we can still use the trained sensor-based HEKM as a stand-alone system to control the robot because the HEKM does not rely on a model of the workspace. The only constraint is to perform the training of the HEKM in a known environment so that the planner can act as a teacher. Then, as we have shown, most of the fine motion knowledge acquired in one environment can be transferred to another one.

This chapter also provides answers to a number of questions concerning the design and properties of the HEKM. *First,* we discussed the utility of using a hierarchical KM instead of the usual "flat" version. The HEKM is more economic in terms of the way memory cells are used. It avoids unnecessary weight repetitions and allows for compact input representations. Clearly, one limitation of the current architecture is the fixed number of neurons. A growing network, able to increase the number of neurons where a more detailed representation is needed, could be used instead [2], [6]. *Second,* we measured the effect of cooperative learning due to the interaction between adjacent neurons. We found that with cooperation learning is slowed down on the short run. But the benefits appear later on, resulting in a more satisfactory final performance. Our interpretation is that, at the beginning of learning, neighboring neurons work to meet a compromise to competing needs: this effort becomes rewarding on the long run. *Third*, we pointed out the complementary nature of the paths generated by the planner and by the HEKM as stand-alone systems. The HEKM produces sub-optimal but smooth solutions, whereas the planner seeks for optimality while sacrificing the continuity of motion. The integration of these two attitudes leads to good results.

Acknowledgments

Cristina Versino was supported by the project No. 21-36365.92 of the Fonds National de la Recherche Scientifique, Bern, Suisse.

References

[1] Clearwater, S.H., Hogg, T., and Huberman, B.A. (1992), "Cooperative problem solving," *Computation: The Micro and the Macro View*, Huberman, B.A. (Ed.), World Scientific.

[2] Fritzke, B. (1995), “A growing neural gas network learns topologies,” *Advances in Neural Information Processing Systems 7*, Tesauro, G., Touretzky, D.S., and Leen, T.K. (Eds.), MIT Press, Cambridge, MA, pp. 625-632.

[3] Gambardella, L.M. and Versino, C. (1994), “Robot motion planning. Integrating planning strategies and learning methods,” *Proc. AIPS94 - The Second International Conference on AI Planning Systems*, Chicago, USA, June.

[4] Gambardella, L.M. and Versino, C. (1994), “Learning high-level navigation strategies from sensor information and planner experience,” *Proc. PerAc94, From Perception to Action Conference*, Lausanne, Switzerland, September 7-9, pp. 428-431.

[5] Heikkonen, J., Koikkalainen, P., and Oja, E. (1993), “Motion behavior learning by self-organization,” *Proc. ICANN93, International Conference on Artificial Neural Networks*, Amsterdam, The Netherlands, September 13-16, pp. 262-267.

[6] Heikkonen, J., Millán, J. del R., and Cuesta, E. (1995), “ Incremental learning from basic reflexes in an autonomous mobile robot,” *Proc. EANN95, International Conference on Engineering Applications of Neural Networks*, Otaniemi, Espoo, Finland, August 21-23, pp. 119-126.

[7] Khatib, O. (1986), “Real-time obstacle avoidance for manipulators and mobile robots,” *The International Journal of Robotics Research*, vol. 5, no. 1.

[8] Knobbe, A. J., Kok, J. N., and Overmars, M.H. (1995), “Robot motion planning in unknown environments using neural networks,” *Proc. ICANN95, International Conference on Artificial Neural Networks*, Paris, France, October 9-13, pp. 375-380.

[9] Kohonen, T. (1984), *Self-Organization and Associative Memory*, Springer Series in Information Sciences, 8, Heidelberg.

[10] Latombe, J.-C. (1991), *Robot Motion Planning*, Kluwer Academic Publishers.

[11] Lozano-Perez, T. (1982), "Automatic planning of manipulator transfer movements," *Robot Motion*, Brady et al. (Eds.), The MIT Press.

[12] Millán, J. del R. (1995), "Reinforcement learning of goal-directed obstacle-avoiding reaction strategies in an autonomous mobile robot," *Robotics and Autonomous Systems*, vol. 15, no. 3, pp. 275-299.

[13] Ritter, H., Martinetz, T., and Schulten, K. (1992), *Neural Computation and Self-Organizing Maps. An Introduction*, Addison-Wesley Publishing Comp.

[14] Ritter, H. and Schulten, K. (1987), "Extending Kohonen's self-organizing mapping algorithm to learn ballistic movements," *Neural Computers*, Eckmiller, R. and von der Marlsburg, E. (Eds.), Springer, Heidelberg.

[15] Versino, C. and Gambardella, L.M. (1996), "Learning fine motion by using the hierarchical extended Kohonen map," in Von der Malsburg, C., Von Seelen, W., Vorbruggen, J.C., and Sendroft, B. (Eds.), *Proc. ICANN96, International Conference on Artificial Neural Networks*, Bochum, Germany, 17-19 July, vol. 1112 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 221-226.