Machine Learning, 8, 363-395 (1992)
© 1992 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

A Reinforcement Connectionist Approach to Robot Path Finding in Non-Maze-Like Environments

JOSÉ DEL R. MILLÁN j_millan@jrc.it
Institute for System Engineering and Informatics, Commission of the European Communities, Joint Research Centre, TP 361, 21020 ISPRA (VA), Italy

CARME TORRAS torras@ic.upc.es
Institut de Cibernètica (CSIC-UPC), Diagonal, 647, 08028 Barcelona, Spain

Abstract. This paper presents a reinforcement connectionist system which finds and learns the suitable situation-action rules so as to generate feasible paths for a point robot in a 2D environment with circular obstacles. The basic reinforcement algorithm is extended with a strategy for discovering stable solution paths. Equipped with this strategy and a powerful codification scheme, the path-finder (i) learns quickly, (ii) deals with continuous-valued inputs and outputs, (iii) exhibits good noise-tolerance and generalization capabilities, (iv) copes with dynamic environments, and (v) solves an instance of the path finding problem with strong performance demands.

Keywords. Connectionism, reinforcement learning, robot path finding, stability, reactive systems

1. Introduction

An important problem in robotics is that of generating a path between initial and goal robot configurations that avoids collisions with obstacles. This is called the *robot path finding problem*. Many attempts at solving the problem have been made in the fields of Artificial Intelligence and Computational Geometry (for reviews, see: Brady, et al., 1982; Whitesides, 1985; Yap, 1987; Torras, 1990). All these approaches are based on *planning*. The time complexity of exact geometric approaches grows exponentially with the number of degrees of freedom of robot motion (Canny, 1988), thus being of practical use only when this number is very low. This fact has led to the emergence of numerous heuristic approaches, which either rely on potential fields (Khatib, 1986) or carry out a search through a state space. These approaches trade reliability for speed, in that they do not guarantee to find a solution when it exists and they can even produce unfeasible solutions because of the use of discretized representations.

There have been some attempts at combining exact and heuristic approaches, so as to exploit their respective strong points and minimize their deficiencies. Along this line, Donald (1987) proposes to carry out a local heuristic search process on an algebraic (exact) model of configuration space,[1] while Ilari and Torras (1990) propose a two-stage process, where first an exact model of *physical space*—i.e., disregarding both the shape and the kinematics of the robot—is used to plan a path and then this path is refined through local search to conform to a *trajectory* in *configuration space*.

The above two-stage decomposition of the path finding problem suggests a possible separation between the path-planning and motion-control aspects of the problem. In *path planning*, the environment is considered at a coarse enough level of detail to be assumed fixed and known a priori—through access to a map—and the goal is to determine, prior to its execution, a sequence of displacements in physical space for reaching a given place. Once a path through physical space is planned at this coarse level, a *high-level motion control* problem arises since commands must be sent to the different motors to make the robot follow the path previously computed, while transforming a discrete path in physical space into a continuous, obstacle-avoiding trajectory in configuration space. Note that, with this problem decomposition, it seems natural to include within motion control the possibility to accommodate for slight deviations in the positions of obstacles, so that it is no longer required that the environment be completely known a priori and static.

In Torras (1990), we have argued that motion planning and motion control involve two different types of processing. Global planning involves reasoning *symbolically* upon an explicit representation of the environment, whilst local obstacle-avoidance capabilities rely on *subsymbolic* pattern processing. This distinction applies to learning as well. Learning to plan would be done symbolically (at a central level), while the learning of reflexes would be carried out at a subsymbolic peripheral level.

This paper focuses on the second aspect above, namely the *subsymbolic learning of obstacle-avoidance reflexes* for an instance of the robot path finding problem characterized by (i) a continuous set of robot configurations and of robot actions, (ii) a partially unknown and dynamic environment, (iii) a need to react in real-time, and (iv) strong performance demands such as finding short paths with wide clearances. Specifically, we present a *reinforcement-based connectionist system* able to generate feasible paths for a mobile robot in a non-maze-like 2D environment, while appropriately dealing with the four problem characteristics above. By saying "non-maze-like" we stress that finding a path in that environment does not require sophisticated planning capabilities, but mainly obstacle-avoidance skills. Maze-like environments require the previous concourse of a path-planner that provides the connectionist system with a series of subgoals along a feasible path, so that the environment around every two consecutive subgoals becomes non-maze-like.

Discovering suitable obstacle-avoidance reflexes by using only a reinforcement signal is a very general approach whose simplest formulation could be characterized as a *weak search method*. This means that reinforcement methods have theoretically limited learning abilities; i.e., they might require heavy learning phases and they might be unable to capture complex features of the problem. These theoretical limitations can be overcome if domain-specific heuristics are incorporated into the basic reinforcement-based search method (Langley, 1985). The *codification scheme* adopted in the present work and the *algorithm used to discover stable solution paths* are instances of such heuristics for the path finding domain. The algorithm allows to greatly *speed up learning* and to deal with *continuous-valued actions*. To the best of our knowledge, this is the first reinforcement system in which continuous-valued actions are used in conjunction with a critic. The codification scheme, besides contributing to solving the problem in a short time, is responsible for the *noise-tolerance* and *generalization* capabilities, for satisfying the *strong performance demands* concerning path length and clearance and, partially, for the ability to cope with *dynamic environments* exhibited by the path-finder.

The paper is structured as follows. In Section 2, previous works relevant to the path-finder developed are reviewed. These are connectionist approaches that either deal with the same particular problem, namely path finding, or tackle the general class of associative reinforcement learning (ARL) problems. The relation of the latter approaches to the system developed becomes clear in Section 3, where path finding is formulated as an ARL problem. Section 4 is devoted to the description of the connectionist path-finder developed. The results of an experimental comparative study of different versions of the general learning rule adopted, as well as the simulation results obtained with the path-finder equipped with the best of such versions are presented next in Section 5. Finally, in Section 6, some conclusions from this work are provided and future work is addressed.

2. Previous work

A general approach to path finding in partially unknown environments is *to build a mapping from perceived situations to correct actions, and iterate this mapping until the goal is reached.* Systems that use this kind of *situation-action rules* are known as *reactive* systems.

Reactive systems normally rely on knowledge-based techniques. However, most of them are not adaptive, i.e., they do not learn (Agre & Chapman, 1987; Arkins, 1987; Schoppers, 1987; Blythe & Mitchell, 1989). In some systems, the situation-action rules are preprogrammed explicitly by their designers (Agre & Chapman, 1987; Arkins, 1987). In other systems, the rules result from a compilation of previous planning activities (Schoppers, 1987; Blythe & Mitchell, 1989). In any case, the main limitation of the knowledge-based systems developed up to now is the need to specify actions to be taken in all possible situations the agent may find itself in.

The knowledge-based reactive systems able to construct by themselves an internal model of the environment have only been applied to simple tasks (Rivest & Schapire, 1987). Mozer and Bachrach (1989) have implemented the Rivest and Schapire's system in a connectionist network, which has been shown to perform better that the original symbolic version in several respects.

Brooks' approach (1986) is a novel way of building reactive systems. Its originality is based on the subsumption architecture and also on hard-wiring sensors to effectors rather than representing the rules. This approach does not overcome the above-mentioned limitation, but it emphasizes a key issue: reactive behavior should involve parallel distributed processing.

2.1. Connectionist approaches to path finding

Subsymbolic systems—connectionist or not—process information in a massively-parallel distributed manner. In Torras (1990) and Millán and Torras (1991a), we stress that subsymbolic techniques should be used to handle the motion control part of robot path finding. Previous subsymbolic attempts to tackle the problem above fall into two categories: constructive and pattern-matching systems.

Constructive path-finders codify the environment through connectionist techniques so as to permit the exploration of configuration space in parallel. Some of them (Steels, 1988) work on a direct discretized representation of the environment, while others capture complex relationships in the environment through recourse to a learning phase (Jorgensen, 1987; Graf & LaLonde, 1988; Mel, 1989). Constructive path-finders, however, suffer from the inherent limitations of needing to carry out a search through a state space, namely off-line processing, static environment and discretization.

Pattern-matching connectionist path-finders are *adaptive* reactive systems, where the mapping from perceived situations to correct actions is learned as paths are generated. They do not need to codify explicitly all the situation-action rules, since they exhibit strong generalization capabilities. However, these systems have only been applied to relatively simple instances of the robot path finding problem. The system Dyna (Sutton, 1990) is the only pattern-matching path-finder that considers the existence of obstacles in the robot's workspace. Although it tackles the path-finding problem as an ARL problem (see the following subsection), as the system presented in this paper does, it has some of the limitations that we would like to overcome. Thus, Dyna relies on a discretization of the workspace, considers a limited set of possible robot steps, and assumes a static environment. In addition, it needs to perform a new learning phase each time the workspace changes.

For a detailed review of these previous connectionist approaches to robot path finding, together with an assessment of their merits and limitations, see Millán and Torras (1991a).

2.2. Connectionist approaches to the ARL problem

The *associative reinforcement learning (ARL)* problem (Barto, et al., 1981; Sutton, 1984; Barto & Anandan, 1985) offers a simple and general framework for developing adaptive reactive systems. Simply stated, the ARL problem is that of learning to associate with each stimulus **X** the action **Y** that maximizes reinforcement z—either present, future or cumulative.

Supervised learning can be used to tackle the ARL problem through the *system identification approach*. As illustrated in Figure 1, this approach consists in training a connectionist

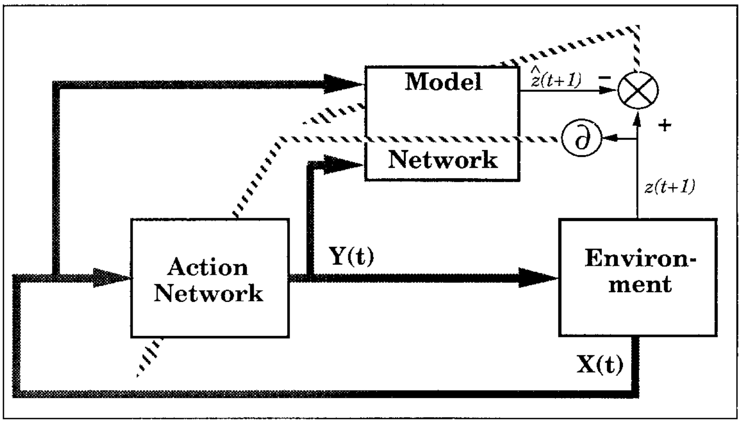


Figure 1. System identification approach to the ARL problem.

network—the *model network*—to identify the function that relates each agents' output with the subsequent reinforcement received from the environment. Once this function is encoded in the model network, the weights of this network are frozen and the supervised learning process shift to the *action network*—i.e., the agent. The gradient of the reinforcement signal with regard to the agent's output, $\partial z/\partial Y$, required for training the action network is estimated by back-propagating derivatives through the model network. This approach has been suggested by Werbos (1987), but the only implementations are those of Munro (1987), Robinson (1989), and Jordan and Jacobs (1990). Although ingenious and analytically well-defined, this approach has the important limitation of assuming that the model has been learned with enough accuracy to allow to compute a good approximation of the reinforcement gradients.

A simpler approach to the ARL problem is *to approximate* $\partial z/\partial Y$ *by its sign*. This idea has been proposed, and successfully applied, by Saerens and Socquet (1989) in the field of control. The signs of the partial derivatives are obtained from qualitative knowledge about the direction in which the action components must be modified to increase the reinforcement signal.

Finally, the most common approach to the ARL problem is *to estimate* $\partial z/\partial Y$ *by measuring the correlation between variations in actions and variations in reinforcement*. This learning paradigm, known as *reinforcement learning*, relies on performing different actions in response to each stimulus, observing the resultant reinforcement, and incorporating the best action into the situation-action rules of the system. Several reinforcement systems have been developed for solving nontrivial problems (Barto, et al., 1983; Anderson, 1987; Chapman & Kaelbling, 1990; Lin, 1990; Mahadevan & Connell, 1990). Anderson (1987) illustrates the kind of situation-action rules learned by a particular reinforcement system.

The two main limitations of the basic reinforcement learning algorithm are a possibly long learning phase and the inability to capture complex features of the problem. To palliate these limitations, several extensions to the basic algorithm have been proposed, such as adding an action model for relaxation planning (Lin, 1990; Sutton, 1990) and combining several modules, each one specialized in solving a particular primitive task (Singh, 1991).

One contribution of this paper is in this direction, since we show that adding a stabilizing strategy to the learning algorithm and incorporating domain-specific knowledge in the codification scheme permits avoiding the above limitations.

Note also that the reinforcement learning approach is open to improvement through the incorporation of supervised learning methods. One classic example of this is the introduction of a *critic* element to predict the amount of reinforcement that would follow a given action (Barto, et al., 1983; Sutton, 1984). Anderson (1986) also used a supervised learning rule for updating the hidden layers of a reinforcement-based system.

3. Path finding as an ARL problem

Because the aim of this work is to carry out a feasibility study on a new approach to robot path finding, we assume a *point robot* and *circular obstacles*. The *evaluation criterion* used is a *compromise between minimizing path length and maximizing the distance to the obstables and walls*. Finally, the goal is considered to have been reached if the current configuration of the robot is below a certain constant distance from the goal, namely $goal_\epsilon$:

$$\text{dist}[conf_{start}, conf_{goal}] < goal_{\epsilon}. \tag{1}$$

To formulate the above instance of the path finding problem as an ARL problem, we need to specify what the input, output and reinforcement signal are.

3.1. Input information

The input to the path-finder consists of an *attraction force* exerted by the goal and several *repulsion forces* exerted by the obstacles. The robot has a limited perception range, giving higher consideration to closer obstacles.

Let the *shortest path vector (SPV)* be the vector that connects the current and the goal robot configurations. The line supporting the SPV and its perpendicular at the current configuration divide the workspace into four quadrants.

The intensity *ia* of the attraction force is an inverse exponential function of the distance x between the current and goal configurations:

$$ia(x) = e^{-k_{att}*x}, \tag{2}$$

where k_{att} is a constant. The direction of the attraction force is that of the SPV, but since the output of the path-finder is computed with respect to this direction (see the next subsection), it need not be included in the input to the system.

Each repulsion force represents the resistance of an obstacle to the fact that the robot follow the SPV. Each such force follows the bisector of the quadrant where the obstacle lies and heads towards the opposite quadrant. For example, in the upper situation in Figure 2, an obstacle is to the southeast of the current configuration and its repulsion force is northwestward. Because the directions of the repulsion forces are specified with respect to the direction of the attraction force, they are implicit in the codification and therefore are not included in the input to the system. Note also that this specification permits grouping

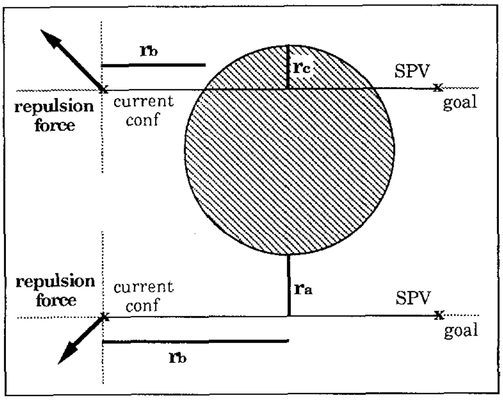


Figure 2. Factors of the repulsion force.

all the repulsion forces into four resulting signals, each being the magnitude of a vector starting at the current configuration and following the bisector of a quadrant.

The intensity of each single repulsion force depends on three factors: (i) the shortest distance between the obstacle and the SPV, r_a, (ii) the distance from the current configuration of the robot to the *point of conflict*—i.e., the point of the SPV nearest to the obstacle—r_b, and (iii) in the case that the SPV intersects the obstacle, the shortest translation of the obstacle perpendicular to the SPV that leads to nonintersection, r_c. If for a configuration and an obstacle several points of conflict exist, then the one nearest to the current robot configuration is taken. Figure 2 illustrates these factors for two different situations and the resulting repulsion forces.

The first factor is aimed at avoiding obstacles in the proximity of the SPV. The second allows to avoid obstacles near to the robot current configuration. The third ensures that, in the case that the SPV intersects an obstacle, the next robot movement is the more distant from the SPV, the deeper is the penetration into the obstacle.

In sum, *the number of input signals* to the path-finder *is independent of the number oj obstacles in the environment*, and these signals are five: the intensity of the attraction force, *ia*, and the *intensity of the environmental repulsion from each quadrant*, r^1, r^2, r^3 and r^4, quadrants being numbered counterclockwise. The detailed way in which the repulsion signals are computed is described in Appendix A. Since the input signals are factors in the learning equations of the system (see Section 4.4), signals with larger magnitudes would have a greater influence on learning than would other signals. To remove this bias all input signals are scaled to lie within the same interval, and are thus codified as real numbers in [0, 1].

It is worth noting that one of the aims of adopting the above codification scheme has been to favor the generalization abilities of the path-finder. Since the goal is codified in the input information and the input is independent of the environment used during the learning phase, the knowledge acquired for a situation should be transferable to a different one.

3.2. Output information

The output of the path-finder represents the step taken by the robot and it is codified as a move in *relative cartesian coordinates* with regard to the SPV. That is, the positive x axis is the SPV. Both increments, Δy and Δx, are real values in the interval [−1, 1].

The output signals determine a direction and a length. Nevertheless, the actual length of the move made by the robot is computed by the following expression:

$$actual\_length = \begin{cases} length * radius, & \text{if } length * radius > step_\epsilon, \\ step_\epsilon, & \text{otherwise,} \end{cases} \tag{3}$$

where $step_\epsilon$ is a constant and

$$radius = \min(perception_{range}, \text{dist}[conf_{start}, conf_{goal}]). \tag{4}$$

The motivation of this postprocessing of the output signals is twofold. Firstly, because the robot concentrates on its neighboring obstacles, the maximum distance the robot can cover in a single step should be limited by the perception range. Otherwise, the robot could collide with obstacles "not completely" perceived. Secondly, the path-finder is intended to be a reactive system, so it should react to each stimulus with some action. $step_\epsilon$ represents the minimum action.

Two important consequences of the manner in which the output signals are codified and postprocessed are that *the space of attainable configurations is continuous* and that *there is no predetermination whatsoever of the direction and length of each step*, with the only constraint imposed by the perception range. A final benefit is that the output postprocessing, in combination with the input codification that supports the generalization capabilities and the reactive nature of the path-finder, offers the robot the possibility of coping with dynamic environments. The step the robot takes is mainly aimed at avoiding neighboring obstacles and it never goes beyond the perception range, therefore the probability of colliding with a mobile obstacle is low.

3.3. Reinforcement signal

The reinforcement signal is a measure of how good is the answer of the system to a particular stimulus. It is calculated on the basis of the quality of the configuration reached by the robot—a combination of its attraction and repulsion factors—and the way in which this configuration has been reached—as measured by the step clearance. The quality of a configuration is an indication of the effort still remaining to reach the goal.

Each robot configuration has two values associated. The first is the *attraction factor*, a, that corresponds to the intensity of the attraction force. The second is the *repulsion factor*, r, that is a function of both the repulsion intensities of the obstacles in front of the robot and the *step clearance*—i.e., the shortest distance to the obstacles and walls of the step with which that configuration is reached:

$$r = \max(r^1, r^4, 1 - step\_clearance). \tag{5}$$

The reinforcement signal, z, is a real number in $[-1, 1]$. It equals 1 when the goal configuration is reached and it equals -1 when a collision with an obstacle or a wall happens:

$$z = \begin{cases} 1, & \text{if } a = 1, \\ -r, & \text{if } r > 1 - rep_\epsilon, \\ a - k_{ar} * r, & \text{otherwise}, \end{cases} \tag{6}$$

where rep_ϵ and k_{ar} are constants.

The present system differs from previous systems tackling the ARL problem in that the latter were intended for single-criterion tasks. In this work we demonstrate that by enriching the reinforcement signal with information that both is available from sensorial data and reflects all the desired criteria (proximity to the goal and clearance), then it is possible to solve problems with *strong performance demands* (reaching the goal quickly and safely).

Table 1. Actual values of the factors and thresholds used for computing the signals.

Name	Value
$goal_{\epsilon}$	0.05
k_{att}	0.1
k_{pa}	100.0
$perception_{range}$	2.0
k_{r_a}	0.2
k_{r_b}	0.1
k_{r_c}	5.0
k_{rep}	1.0
$step_{\epsilon}$	0.01
k_{ar}	0.75
rep_{ϵ}	0.1

Finally, the way in which the reinforcement signal is computed allows to reduce the complexity of the task to be solved. In the robot path finding problem, the consequences of an action can emerge later in time. Thus, actions must be selected based on both their short- and long-term consequences. Since the reinforcement signal is computed using *global information*—i.e., it is based not on the current robot configuration but on the SPV—the path-finder gets a measure of the short- and long-term consequences of an action only one time-step after executing it. Thus the task is reduced to learn, for each stimulus, to perform the action which maximizes the reinforcement signal.[2]

The determination of both the input and the reinforcement signals is reminiscent of the *potential field approach* to path finding (Khatib, 1986).

Table 1 summarizes all the constants—coefficients and thresholds—and their actual values used to compute the different signals.

4. A reinforcement connectionist path-finder

The path-finder is made of two elements, namely the *step generator* and the *critic*, and interacts with its environment as depicted in Figure 3. At each time t, the environment provides the path-finder with the input pattern $\mathbf{X}(t) = (x_1(t), x_2(t), x_3(t), x_4(t), x_5(t))$, together with the *environmental reinforcement signal* $z(t)$. The input pattern is fed to both the step generator and the critic. Nevertheless, the step generator does not receive directly the environmental reinforcement signal but the *heuristic reinforcement signal* $h(t)$ elaborated by the critic. The latter is an enrichment of the former based on past experience of the path-finder when interacting with the environment. The step generator produces instantaneously an output pattern $\mathbf{Y}(t) = (y_1(t), y_2(t))$ that it is the output of the path-finder. The environment receives this action $\mathbf{Y}(t)$ and, at time $t + 1$, sends to the path-finder both an evaluation $z(t + 1)$ of the appropriateness of the action $\mathbf{Y}(t)$ for the stimulus $\mathbf{X}(t)$ and a new stimulus $\mathbf{X}(t + 1)$.

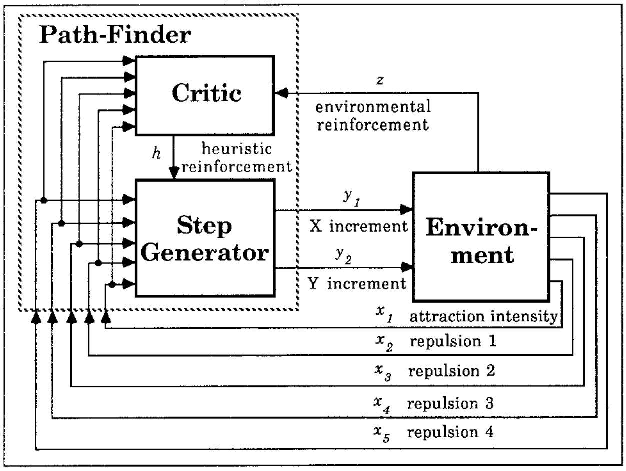


Figure 3. Connectionist path-finder according to the ARL architecture.

Of the two kinds of reinforcement learning algorithms proposed to solve the ARL problem, namely *associative reward-penalty*, A_{R-P}, (Barto & Anandan, 1985) and *associative search*, *AS*, (Barto, et al., 1981; Sutton, 1984; Williams, 1986, 1987), we adopt the second one because it fits better the specifications of the output and reinforcement signals we have. The A_{R-P} algorithm has been mainly designed for instances of the ARL problem where the action network—the step generator in our case—produces binary output signals and where the reinforcement has only two possible values, namely success or failure.

A central issue for any reinforcement system is to explore alternative actions for the same stimulus. *Stochastic units* provide this source of variation. Thus, the step generator—or, at least, its output layer—is built out of this kind of unit. The different architectures of the step generator—i.e., number of hidden layers, kinds of hidden units, and connectivity—tested during the simulations will be described in the next section. Nevertheless, the stochastic behavior of the path-finder should tend to be *deterministic* with learning. Otherwise, the path-finder could not generate *stable solution paths* after it eventually discovers them.

4.1. The basic AS algorithm

Continuous stochastic units compute their output in three steps (Gullapalli, 1988), as depicted in Figure 4 and expressed in Equations (7) through (10).

Since the signals we are interested in are continuous, a separate control of the location being sought (*mean*) and the breadth of the search around that location (*variance*) is needed. The first step is to determine the value of these parameters. The mean μ should be an estimation of the optimal output. A simple way is to let $\mu_i(t)$ equal a weighted sum of the inputs $s_j(t)$ to the unit i plus a threshold $\theta_i(t)$:

$$\mu_i(t) = \sum_{j=1}^{n} (w_{ij}(t)s_j(t)) + \theta_i(t). \tag{7}$$

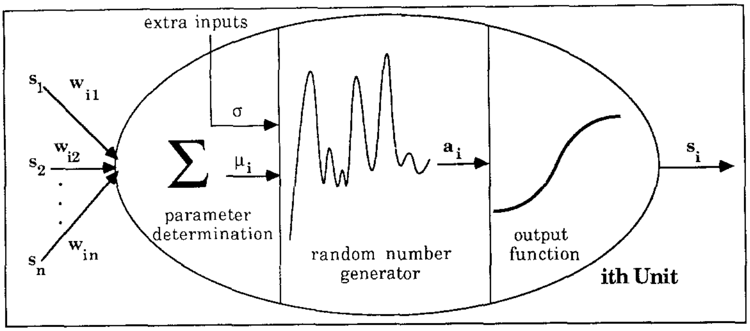


Figure 4. Structure and functionality of a stochastic unit.

The standard deviation σ should be small if the expected output of the step generator is close to the optimal, and it should be high in the opposite case. Since the heuristic reinforcement signal provides this comparative measure of output goodness, $\sigma(t)$ should depend on the *expected heuristic reinforcement*, $\hat{h}(t)$:

$$\sigma(t) = g(\hat{h}(t)), \tag{8}$$

where the function g will be made explicit in Section 4.3, once the different ways of computing the heuristic reinforcement signal will be presented.

As a second step, the unit calculates its *activation level* $a_i(t)$ which is a normally distributed random variable:

$$a_i(t) = N(\mu_i(t), \sigma(t)). \tag{9}$$

Finally, the unit computes its output $s_i(t)$:

$$s_i(t) = f(a_i(t)) = \frac{2}{1 + e^{-\beta a_i(t)}} - 1, \tag{10}$$

where β is a constant in [0, 1].

In the AS family of algorithms, the weights are modified according to the following general expression:

$$\Delta w_{ij}(t) = \alpha h(t) e_{ij}(t - 1), \tag{11}$$

where α is the *learning rate* and e_{ij} is the *eligibility factor* of w_{ij}. The eligibility factor of a given weight measures how influential that weight was in choosing the action. We have explored twenty-five versions of this general rule, each particular version differing from the others in the way h and e_{ij} are calculated. These versions result from combining the five heuristic reinforcement signals with the five eligibility factors which shall be presented in Sections 4.3 and 4.4.

4.2. A strategy for discovering stable (quasi) optimal paths

It has been stated above that in order to obtain stable solution paths, stochastic units should become deterministic as learning proceeds. The way in which σ is computed guarantees that the breadth of search will diminish asymptotically to zero as the path-finder learns. But this is not acceptable to solve the problem efficiently.

We have extended the basic AS algorithm with a mechanism for accelerating the generation of stable solution paths (Millán & Torras, 1990). When the path-finder, after a certain experience with the environment, discovers an *acceptable* path, the search for a solution path proceeds in a more "controlled" manner by *transforming the stochastic units of the step generator into deterministic units*. In addition, the weights are not updated after each step, but after the path has been generated and only if it is not a *(quasi) optimal* path, *qo-path*. The acceptability and optimality criteria are defined by the boolean expressions:

$$acc\_path = (length < k_{acc_{length}} * \mathrm{dist}[conf_{start}, conf_{goal}]) \wedge (minimum\_clearance < k_{acc_{clear}}), \tag{12}$$

$$qo\_path = (length < k_{qo_{length}} * \mathrm{dist}[conf_{start}, conf_{goal}]) \wedge (minimum\_clearance < k_{qo_{clear}}), \tag{13}$$

where $k_{acc_{length}}$, $k_{acc_{clear}}$, $k_{qo_{length}}$ and $k_{qo_{clear}}$ are workspace-dependent constants.

A deterministic unit is like a stochastic one but without the random component: the activation level is a weighted sum of the input. The weights arriving at a deterministic unit are modified using the same learning rule as for the weights arriving at stochastic units.

The weight updates are delayed and not applied when a qo-path is generated because, otherwise, the changes to the weights could probably prevent the path-finder from reproducing it. Since the output signals are continuous and the optimality criterion is very severe, even little weight modifications could alter the step generator so as to produce actions sufficiently away from the quasi-optimal ones.

The process for discovering stable qo-paths requires a further refinement. The fact of obtaining an acceptable path does not necessarily imply that a qo-path is nearby. The acceptable path discovered could be around a local optimum. Thus, if after consuming a fixed quantity of computational resources the path-finder does not discover a qo-path, then the deterministic units are turned back to being stochastic again and another acceptable path is looked for. Specifically, the deterministic phase is finished when either a single path reaches a certain number of steps or a given number of paths are generated without finding a qo-path.

The strategy above is sufficient for finding a stable qo-path. Nevertheless, we have found that sometimes the trajectory followed by the weight configuration of the step generator reaches bad states. The algorithm eventually allows to escape from them, but at the cost of spending a lot of time. This time could be significantly reduced if these states were detected as quickly as possible and a new state (at least as good as the initial one) was readily imposed. An easy way of implementing this idea is not to allow the path-finder

```
Algorithm discover_stable_qo-path
begin
    repeat
        $counter := 0$
        initializations
        repeat
            repeat
                generate_path (immediate_learning, stochastic_units)
            until (acceptable_path and enough_experience)
            repeat
                generate_path (delayed_learning, deterministic_units)
            until (qo-path or no_more_resources)
            $counter := counter + 1$
        until (qo-path or limit_of_iterations)
    until qo-path
end
```

Figure 5. Strategy for discovering a stable qo-solution.

to alternate the deterministic and stochastic phases more than a fixed number of times. If this limit is reached, the step generator and the critic are reinitialized.

Figure 5 shows the final algorithm for discovering a stable qo-path.

4.3. The critic

As stated above, the goal of the critic is to transform the environmental reinforcement signal into a more informative signal, namely the heuristic reinforcement signal. This improvement is based on past experience of the path-finder when interacting with the environment, as represented by the *reinforced baseline b*:

$$h(t) = z(t) - b(t - 1). \tag{14}$$

The critic receives the input pattern $\mathbf{X}(t)$ and predicts the reinforcement baseline $b(t)$ with which to compare the associated environment reinforcement signal $z(t + 1)$.

A first alternative for the reinforcement baseline is to make it a *constant*—zero in the simulations presented here. This is equivalent to shutting the critic off. One has to expect a long learning phase before obtaining a qo-path, since the robot may receive a positive environmental reinforcement when moving away from the goal, in cases where it can approach it.

In order to avoid this shortcoming, the environmental reinforcement received at the current step should be compared with that received at previous steps. Two possibilities for making this comparison arise: *short-term*, and *long-term*. In the first case, the reinforcement baseline is the environmental reinforcement received at the preceding time step. In the second case, it is a trace of all the environmental reinforcement received by the path-finder. There are, however, reasons to doubt that these techniques would work well on the task at hand. The main of them is that the best move the robot can make in a certain situation may correspond an environmental reinforcement lower than the preceding one.

A third comparison technique, *predicted comparison* (Sutton, 1984), tries to overcome this limitation by computing the reinforcement baseline as a prediction based on the environmental reinforcement received when the same—or similar—input patterns occurred. That is, the critic has to predict the environmental reinforcement signal $z(t + 1)$ to be received by the path-finder when the stimulus $\mathbf{X}(t)$ is present. In order to undertake this task, the critic is built as a second network out of deterministic units, and since it is provided with input/output pairs, a supervised learning algorithm is used to learn to associate each stimulus with the corresponding environmental reinforcement signal. In particular, we have used the "on-line" version of the backpropagation algorithm with a *momentum* term:

$$\Delta v_{ij}(t) = -\epsilon \frac{\partial(z(t) - \hat{z}(t - 1))^2/2}{\partial v_{ij}} + \eta \Delta v_{ij}(t - 1), \tag{15}$$

where $\mathbf{V}$ is the weight matrix of the critic, ϵ is the learning rate, η is the momentum factor, and $\hat{z}$ is the output of the critic—i.e., the expected environmental reinforcement signal.

The critic has to be updated after each path-finder/environment interaction because as the step generator improves, the mapping from stimuli to reinforcement changes.

We have found that using a large learning rate ϵ for the critic and a small learning rate α for the step generator accelerates the learning process. The rationale is the following. By allowing the critic to adapt more rapidly than the step generator, the critic has the opportunity to predict acceptably the next environmental reinforcement signal associated to the current stimulus as the step generator is, during a certain period of time, almost stable. Then, the step generator can take full advantage of the reinforcement baseline.

A potential benefit of the momentum term when applied to *dynamic tasks*—i.e., tasks where the training information changes over time—like that faced by the critic is that it prevents $\mathbf{V}$ from oscillating dramatically. Since, at the beginning of the learning phase, the step generator explores a wide range of alternative actions, to each stimulus perceived by the path-finder will correspond a large variety of environmental reinforcement signals. Consequently, the error between the actual and the predicted environmental reinforcement is not only due to a misfunction of the critic, but also to the "chaotic" mapping the critic is dealing with. So, there is the risk that the weight modifications made at a certain time change $\mathbf{V}$ too much. The momentum tackles this problem by reducing the intensity of opposite weight modifications at consecutive times.

A second advantage of using a predicted-comparison reinforcement baseline is to mitigate the possibility of *overlearning*. Since the goal of the step generator is to produce the optimal action in the presence of every stimulus, learning should stop when the step generator has discovered the suitable situation-action rules. A predicted-comparison mechanism accomplishes this effect because the expected and the actual environmental reinforcement signals tend to be the actual and the optimal ones, respectively, as the critic and the step generator improve.

Predicted comparison, however, may not cope adequately with collisions—a frequent situation during the learning phase. The reason is that when a collision happens, one would like to punish severely the step generator. This is satisfied if no baseline is used, whereas predicted comparison fails when the prediction is close to the environmental reinforcement signal. Consequently, the performance of the path-finder should improve if the reinforcement baseline were computed as a *heuristic predicted comparison*:

$$\tilde{z}(t) = \begin{cases} 0, & \text{if } z(t+1) = -1, \\ \hat{z}(t), & \text{otherwise} \end{cases} \tag{16}$$

The following expression summarizes the five alternative ways used to calculate the reinforcement baseline:

$$b(t) = \begin{cases} 0, \\ z(t), \\ bt(t), \\ \hat{z}(t), \\ \tilde{z}(t), \end{cases} \tag{17}$$

where

$$bt(t) = \lambda z(t) + [1 - \lambda]bt(t-1), \tag{18}$$

with λ being a constant in [0, 1].

Let us now specify the function g used to calculate the standard deviation σ of the stochastic units in the step generator. Remember that σ should be small if the expected output of the step generator is close to optimal, and it should be high in the opposite case. The last four ways of computing the reinforcement baseline lead to a heuristic reinforcement signal whose absolute value is close to zero if the environmental reinforcement is close to the expected one, and it is high in the opposite case. Consequently, $\sigma(t)$ should be proportional to the expected heuristic reinforcement $\hat{h}(t)$:

$$\sigma(t) = k_\sigma * \hat{h}(t), \tag{19}$$

where k_σ is a constant and $\hat{h}(t)$ is a trace of the absolute value of past heuristic reinforcement received:

$$\hat{h}(t) = \xi * \text{abs}(h(t)) + [1 - \xi]\hat{h}(t-1), \tag{20}$$

ξ being a constant in [0, 1]. Gullapalli proposes the following expression to compute σ:

$$\sigma(t) = k_\sigma * [1 - \hat{z}(t)], \tag{19'}$$

where $\hat{z}(t)$ is the expected environmental reinforcement. Both expressions are equivalent only in the case that the highest environmental reinforcement is recieved for every pair (stimulus, optimal action). This condition does not hold in the path finding problem where the highest environmental reinforcement is only received when the goal is reached.

In the case that no reinforcement baseline is used, the heuristic reinforcement is the environmental one. This means that (19) provides a standard deviation that is not the desired one when the expected output is good, as σ will be high when it should be small. For this particular case, (19′) will be used.

4.4. *The eligibility factor*

The eligibility factor of a weight measures the contribution of that weight to the action taken. We have used two kinds of rules to compute this contribution. The first measures the correlation between the outputs of the units linked by the connection under consideration. Two ways of implementing this *intensity-based* mechanism are the *Hebbian rule* and a trace of it.

The second kind of rule consists of a *discrepancy-based* mechanism which evaluates the difference between the actual output and the expected one:

$$e_{ij}(t) = s_j(t)[s_i(t) - f(\mu_i(t))], \tag{21}$$

where $f(\cdot)$ is the same function as in (10). A slightly different implementation proposed by Gullapalli (1988) is the following:

$$e_{ij}(t) = s_j(t) \frac{a_i(t) - \mu_i(t)}{\sigma(t)}. \tag{22}$$

Finally, a third discrepancy-based rule makes the learning algorithm perform *gradient ascent* on the expected environmental reinforcement, as proved by Williams (1986, 1987):

$$e_{ij}(t) = \frac{\partial lnN}{\partial w_{ij}}(t) = s_j(t) \frac{a_i(t) - \mu_i(t)}{\sigma^2(t)}, \tag{23}$$

where N is the normal distribution function in (9).

In the three cases, the greater the discrepancy between the actual and the expected outputs, the greater the intensity of the weight modification. The sign of this modification is such that if the heuristic reinforcement signal is positive, then the next time the current stimulus—or similar ones—will be presented, the expected output of the stochastic unit will be closer to the current one. Conversely, if the heuristic reinforcement signal is negative, then the weights are modified so as to reduce the probability of generating the current output.

Sutton (1984) proved experimentally that such a mechanism allows a binary stochastic unit to discriminate between similar input vectors, and Barto (1985) argued that it helps to cope with situations where two or more of the actions the path-finder can take in the presence of a given stimulus have associated environmental reinforcement signals of similar intensity.

The following expression summarizes the five alternative ways in which the eligibility factor has been calculated:

$$e_{ij}(t) = \begin{cases} s_i(t)s_j(t), \\ et_{ij}(t), \\ s_j(t)[s_i(t) - f(\mu_i(t))], \\ s_j(t)\,\dfrac{a_i(t) - \mu_i(t)}{\sigma(t)}, \\ s_j(t)\,\dfrac{a_i(t) - \mu_i(t)}{\sigma^2(t)}, \end{cases} \tag{24}$$

where

$$et_{ij}(t) = \gamma s_j(t)s_i(t) + [1 - \gamma]et_{ij}(t - 1), \tag{25}$$

γ being a constant in [0, 1].

Note that only the first two rules—i.e., those corresponding to the intensity-based mechanism—are applicable when deterministic units are in use. The other three eligibility factors depend on the stochastic behavior of the units.

5. Experimental results

The system has been implemented in Common Lisp on a VAX station 2000. The feasibility study has been carried out in two main steps. The first step is an experimental comparison of the twenty five versions of the general AS algorithm described in the preceding section. The objective of this comparative study is to identify the version that best suits the robot path finding problem. Then, the most promising version is used to try to build a path-finder with powerful generalization abilities so as to produce, after a limited learning process, qo-paths in any non-maze-like workspace and to cope with dynamic environments.

The workspace depicted in Figure 6 has been employed in the simulations. There exist three obstacles—shown as circles—and the goal configuration is represented by a square. The workspace is a room of 10 $*$ 10 units. The centers of the obstacles are located at (3.0, 6.0), (6.0, 6.0) and (6.0, 3.0), and their radii are equal to 0.75 units. The goal configuration is located at (8.0, 8.0).

In the simulations, a path is considered to end either at the goal or with a collision. A path could be alternatively defined to finish only when the goal is reached. This second definition, however, presents the following shortcoming. During the learning phase, the path-finder needs to be *sufficiently stimulated*—i.e., the training data should be both sufficiently varied to reveal the reinforcement function and sufficiently repetitive to make learning possible—to discover the set of suitable situation-action rules. But, if the robot is allowed to leave the collision situations by itself, then the proportion of inputs corresponding to a collision with respect to all the other kinds of inputs is very high. So, the training data is biased to a particular kind of stimuli, which makes learning harder.

A desirable feature for any system is its robustness to changes in the values of the parameters governing it. Thus, no attempt has been made to search for the best set of parameter

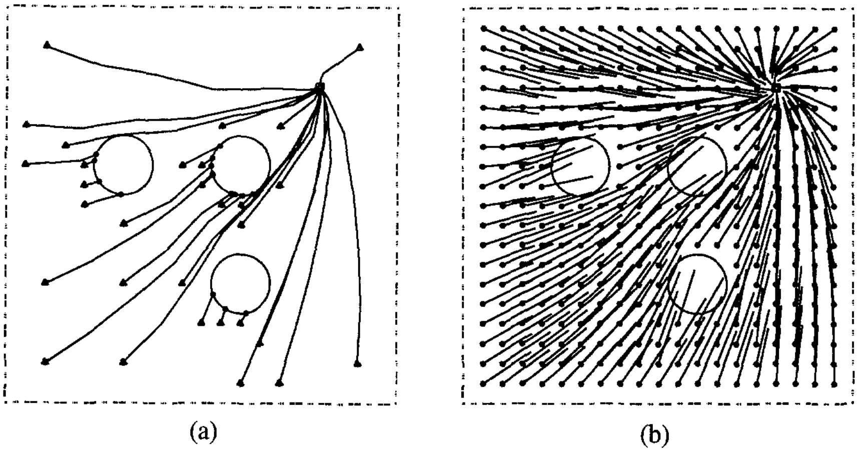


Figure 6. Behavior of the first path-finder.

values. Instead, we have chosen values that intuitively seem adequate for all the versions of the general AS rule. The values of the constants used in the learning rules appear in Table 2.

In the implementation of the strategy described in Section 4.2, the deterministic phase is finished when either a single path reaches 500 steps or 500 paths are generated without finding a qo-path.

Table 2. Actual values of the factors used in the learning rules.

Name	Value
β	0.4
α	0.125
ϵ	0.5
η	0.5
λ	0.2
γ	0.2
k_σ	2.0
ξ	0.75
$k_{acc_{length}}$	1.5
$k_{acc_{clear}}$	0.15
$k_{qo_{length}}$	1.15
$k_{qo_{clear}}$	0.2

5.1. Tuning the learning rule: A comparative study

The step generator used to carry out the comparative study has only two layers of units, input and output. The critic has a hidden layer of four units.

The following computer experiment has been designed. The initial configuration is located at (3.0, 3.0), in such a manner that the obstacles lie in between it and the goal. Thus the path-finder is required to produce a nontrivial qo-path. A run finishes when a qo-path is discovered.

For each version, twenty qo-paths have been generated. Before beginning the generation of each qo-path, all the weights of the step generator are set to zero and the weights of the critic are initialized randomly.

Given the difficulty of generating acceptable paths, the experience acquired by the path-finder for producing the first acceptable path is considered sufficient to finish the stochastic phase of the algorithm.

Versions using a discrepancy-based rule for computing the eligibility factor turn to using the Hebbian rule when units become deterministic (see the next subsection for a justification of this choice).

Table 3 gives the average number of steps required to generate a stable qo-path for every version. The types of reinforcement baseline and eligibility factor are numbered according to their positions in (17) and (24), respectively. The results of these experiments show that every version, if given enough time, produces a stable qo-path from the initial to the goal configurations. Nevertheless, important differences exist in the performance of the various versions.

5.1.1. Eligibility factor

Let us firstly concentrate on the role of the eligibility factor in the performance of the path-finder. Three main conclusions can be extracted from the data in Table 3.

First, discrepancy-based rules (three last columns in the table) clearly outperform intensity-based ones.

The second conclusion is that the performance of the versions adopting the Hebbian rule—first column—is always better than the performance of the versions using a trace of this rule—second column. This is the reason for using the Hebbian rule in the deterministic phase, as mentioned in the preceding subsection.

Table 3. Average number of steps for finding a stable qo-path.

		Eligibility				
		1	2	3	4	5
Baseline	1	34939	46122	15559	14527	10457
	2	59599	66791	48753	24810	32332
	3	31870	52202	28379	17494	27645
	4	25997	27999	12675	13839	08473
	5	21362	22840	16301	11789	17907

Finally, of the three discrepancy-based rules, none outperforms the others for all choices of reinforcement baseline. However, the fifth eligibility factor leads to the two best versions of the algorithm.

5.1.2. Reinforcement baseline

The influence of the reinforcement baseline on the performance of the path-finder is not so clear and does not agree so exactly with the theoretical expectations as the influence of the eligibility factor.

As expected, short-term and long-term comparisons—second and third rows of the table, respectively—are not suitable for solving our formulation of the path finding problem. In addition, the long-term versions work better than the short-term ones.

Surprisingly, the performance of the versions adopting a null reinforcement baseline—first row—is very similar to the performance of the versions using predicted comparison as reinforcement baseline—fourth row—when combined with discrepancy-based eligibility factors. Nevertheless, the former never outperforms the latter. An explanation for this fact could be that a null reinforcement baseline deals better with collisions than predicted comparison, as we hypothesized in Section 4.3. However, the performance of the versions using heuristic predicted comparison—fifth row—which combines a null reinforcement baseline and predicted comparison, does not support this hypothesis, since it is considerably worse than the performance of plain predicted comparison when two of the three best eligibility factors are used—third and fifth columns.

So, it seems that predicted comparison is the best kind of reinforcement baseline.

5.1.3. Conclusions

Since the performance of the versions combining the first, fourth and fifth types of reinforcement baseline—i.e., null, predicted comparison and heuristic predicted comparison, respectively—with the third, fourth and fifth types of eligibility factors—i.e., the three discrepancy-based rules—are all quite similar, we will look at some other information to better discriminate which of these nine versions is the best.

One of these criteria is the average number of times that the path-finder has been reinitialized until a stable qo-path is found. This information is implicit in Table 3, as all the steps taken by the robot for finding a stable qo-path are recorded. Nevertheless, this information provides a picture of the sensitivity of each version to the initial conditions. Clearly, the less a version depends on its initial conditions, the more it should be preferred. Table 4 shows the average number of reinitializations of the nine most promising versions identified before.

Table 4 confirms the ranking we have outlined in the preceding subsections. The reinforcement baseline and the eligibility factor most suitable for our formulation of the robot path finding problem are predicted comparison and the discrepancy-based rule proposed by Williams (1986), respectively. This version, however, is not far better than some other versions.

Appendix B provides an analysis of variance for the most promising versions that supports these conclusions.

Table 4. Average number of reinitializations of the most promising versions.

		Eligibility		
		3	4	5
Baseline	1	1.5	1.4	1.1
	4	1.4	1.4	0.9
	5	1.6	1.3	1.5

5.2. Configuring the path-finder

The second phase of the feasibility study is aimed at building a path-finder with powerful *generalization abilities* which, after a limited learning process, produces qo-paths in any non-maze-like workspace and is able to cope with dynamic environments.

The workspace and the goal used in the experiments are the same as before. The training set consists of a certain number of initial configurations that are representative of the whole workspace. Because of the symmetry of this workspace, the training set consists of pairs of symmetrical configurations.

Training is carried out in an incremental way, using an extension of the algorithm shown in Figure 5. That is, the path-finder has to learn to generate a qo-path from the first starting configuration in the training set. Then, it tries to generate a qo-path from the second starting configuration. As the necessary knowledge required for solving this new situation is discovered and codified, part of the previous knowledge could be lost. So, the path-finder must learn again to generate a qo-path from the first starting configuration, but this time using directly deterministic units. The iteration "learn to generate a qo-path from the new starting configuration—recover the ability to generate qo-paths from the previous starting configurations" is repeated until the path-finder is able to generate qo-paths from all the starting configurations.

The fact that the path-finder deals with the starting configurations one at a time is a constraint imposed by the stability strategy.

To verify if it is still able to generate qo-paths from a previous starting configuration, the path-finder is made to produce a path, without modifying the critic and step generator networks, and the optimality criterion is applied. The steps in this path are not counted in the performance records.

All the system parameters but two have the values appearing in Table 2. The two parameters modified are k_σ and $k_{qo_{length}}$. In the simulations reported in this section, $k_\sigma = 1.0$ and the following distinction has been introduced with regard to $k_{qo_{length}}$. The path-finder has to deal now with two kinds of initial configurations, namely those such that there is no obstacle in between them and the goal and those such that there are. For the first group of initial configurations, the path-finder should generate qo-paths as straight as possible. So, a stronger optimality criterion has been defined by choosing a smaller $k_{qo_{length}}$. This value is 1.05. For the second kind of initial configurations, $k_{qo_{length}}$ continues to be 1.15.

5.2.1. Problem non-linearity

The first path-finder we have experimented with is the very simple one used to carry out the comparative study. That is, the step generator does not have any hidden layer of units and the critic has four hidden units arranged in a layer.

Unfortunately, the path-finder is not able to tackle satisfactorily the problem at hand. The reason is that, for our formulation of the robot path finding problem, *nonlinear associations* between the input and output signals are required. An illustration of one of these nonlinear associations follows.

Let us assume that there is no obstacle behind the current configuration of the robot, that is, $r^2 = r^3 = 0$. Let us consider now how the output of the Δx unit should be for different obstacle situations. Since the current and goal configurations are fixed, the intensity of the attraction force is constant. So, we will only consider the values of Δx for different instances of the pair (r^1, r^4). Figure 7 depicts the sign of Δx when the values r^1 and r^4 lie in three different ranges, namely *low*, *medium* and *high*.

The robot tries normally to approach the goal—i.e., Δx is positive. The only exception is when there are obstacles at the two sides of the SPV and they are close to the current configuration. The figure shows that the two regions of the space, one where Δx is positive and the other where Δx is negative, are not linearly separable.

Even though the simple path-finder used in the comparative study does not solve completely the instance of the path finding problem we are interested in, it illustrates the potentiality of a reinforcement learning connectionist approach. Figure 6 shows the behavior of the path-finder after knowing how to produce qo-paths from a particular set of starting configurations, namely those such that there is no obstacle in between them and the goal. Panel A depicts the paths generated by the path-finder from every starting configuration in the training set. Each initial configuration is represented by a triangle. In panel B, instances of the situation-action rules discovered by the path-finder are illustrated; for every starting

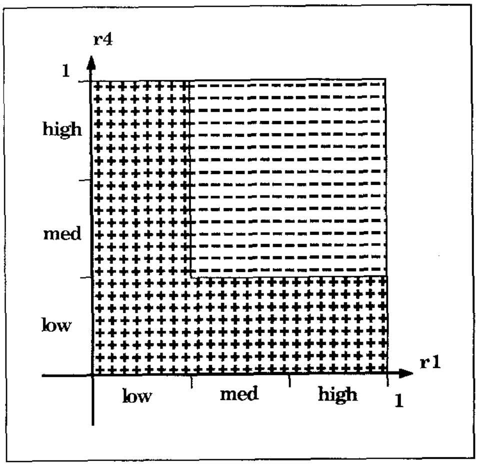


Figure 7. Nonlinearity of the mapping $(r^1, r^4) \rightarrow \Delta x$.

configuration considered—little circles—the move to be taken by the robot is shown. From the figure it is evident that the path-finder is able to tackle many more situations than those perceived during the learning phase. But, the most relevant feature is that *this simple path-finder knows the right direction, left or right, for the moves to avoid obstacles.* The cost of generating this particular path-finder has been 515 steps.

5.2.2. System performance

Given the abilities of this simple path-finder, we would like to add a hidden layer to the step generator in such a manner that these abilities are preserved when the missing nonlinear constraints are incorporated and the situation-action rules already discovered are tuned. To this end, the second path-finder we have tried has the same critic above, and its step generator is a network where all the units in a layer are connected to all the units in all the layers above. The weights associated to the direct links between the input and output layers are those of the first step generator and are not modified during the second learning phase. The hidden layer of the step generator is made of four units. The weights of the critic are initialized to the weights resulting from the first phase.

During this second learning phase, the new path-finder is made to deal with those starting configurations that have obstacles in between them and the goal, following the same incremental strategy as above with regard to previously learned situations. Besides, the path-finder is considered to have the necessary experience with the environment when it has generated three acceptable paths.

The purpose of the hidden layer is to encode the missing nonlinear features of the robot path finding problem. This layer can be built according to an *extension-and-homogeneity (EH)* principle or to a *modular-and-functionality (MF)* principle. If the first principle is adopted, the resulting step generator is totally made up of stochastic/deterministic units following the same version of the AS learning rule.

Anderson (1986) has found that this approach to the ARL problem leads to an inefficient learning process. He overcomes this limitation by adopting the second principle above. That is, the step generator is broken down into two modules—input-hidden layers and hidden-output layers—and the most suitable learning algorithm for each module's functionality is applied. In particular, Anderson developed an *approximate gradient method* for discovering the internal representations required. This method is an adaptation of the backpropagation algorithm where the error back-propagated from the i^{th} output unit to the j^{th} hidden unit at time t is $h(t)e_{ij}(t-1)/s_j(t-1)$.

Formally, if Anderson's version of the MF principle is adopted, the step generator is made of two modules. The first module consists of the output units and the connections arriving at them. The output units are stochastic/deterministic and the AS learning rule is used to update the connections. The second module consists of the hidden units and the connections arriving at them. The hidden units are deterministic and the weights associated to those connections are updated according to the expression:

$$\Delta w_{jk}(t) = \alpha_h \delta_j(t) s_k(t-1), \tag{26}$$

where α_h is the learning rate and

$$\delta_j(t) = \sum_{i=1}^{2} \left[\frac{h(t)e_{ij}(t-1)}{s_j(t-1)} w_{ij}(t-1) \right] s_j(t-1)[1 - s_j(t-1)], \qquad (27)$$

where k ranges over the input units, j over the hidden units, and i over the output units.

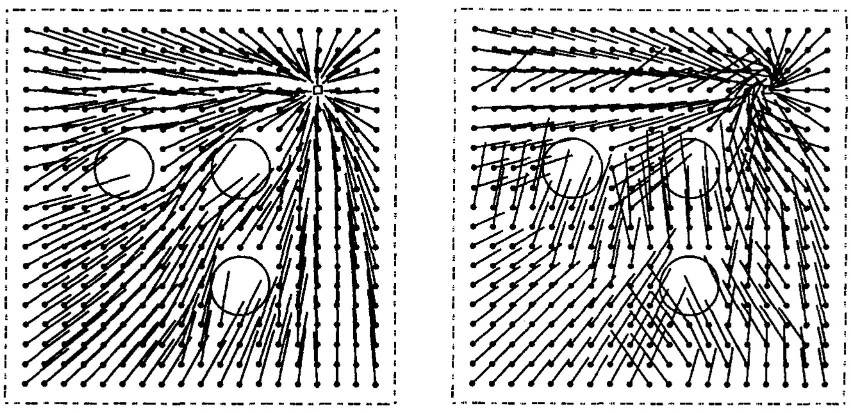
We have compared these two principles on the generation of qo-paths for a pair of symmetrical initial configurations that have obstacles in between them and the goal. In the simulations, $\alpha_h = 0.125$, therefore the learning rate is the same for all the units of the step generator. For each principle, five qo-paths have been generated. The average number of steps taken for networks adopting each principle appear in Table 5.

The computer experiments confirm that the performance of the MF principle is better than that of the EH principle. Nevertheless, the difference is negligible, perhaps because the task is not sufficiently difficult and, above all, because the system does not begin from scratch but with the "approximate" desired knowledge.

The most important result of this comparative study is illustrated in Figure 8. Panel A and panel B depict the behavior of two of the path-finders generated, one using the EH principle and the other the MF principle. Similar behaviors have been observed for other path-finders using these principles. The unexpected feature is that *the EH principle is able to modify correctly all the situation-action rules already discovered*, while *the MF principle is not.*

Table 5. Average number of steps required by each principle.

Principle	Steps
EH	3487
MF	3307



(a) (b)

Figure 8. Behavior of two path-finders adopting each principle.

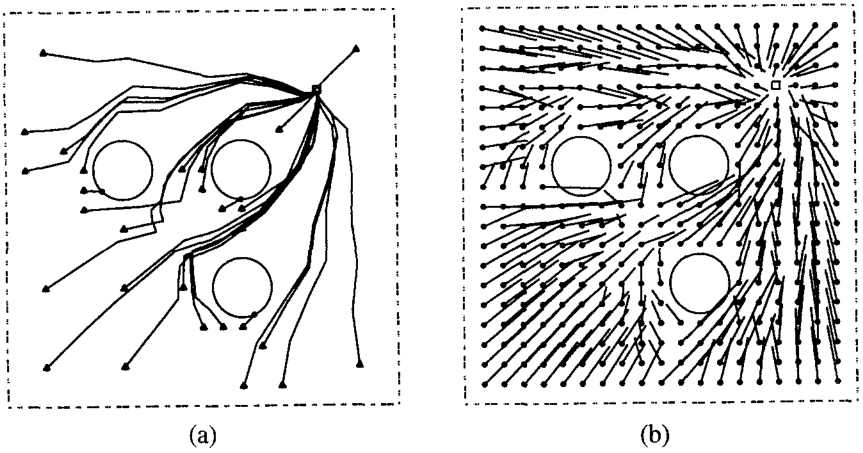


Figure 9. Behavior of the second path-finder.

Figure 9 shows the behavior of a pure reinforcement path-finder (EH principle) that is able to produce qo-paths from almost all the initial configurations in the training set. The number of steps required to reach this state of the path-finder has been 77315. The only situations not properly handled by the path-finder are a subset of the most difficult ones, that is those in which an obstacle lies in between the goal and the current robot configuration and is very close to the latter. These situations may be handled by getting appropriate guidance from a symbolic path-planner (Millán & Torras, 1991b).

It is worth noting that no two-layered step generator trained in one phase succeeded.

5.2.3. Noise-tolerance, generalization and dynamic capabilities

Until now, we have assumed that the robot can perceive the workspace perfectly. Nevertheless, a robot navigating in a real environment is subject to noisy and inaccurate measurements. Figure 10 depicts the behavior of the path-finder when sensory data are disrupted with 20% of white noise. In panel A, the noise is added to each input signal of the path-finder—i.e., r^1, r^2, r^3, r^4 and *ia*. In panel B, it is added to the basic sensory input of the robot—i.e., r_a, r_b, r_c and *ia*. Comparing the two panels of Figure 10 with Figure 9, panel B, it is evident that the path-finder exhibits a *large noise tolerance*, since the "maps" are very similar. The behavior of the path-finder is slightly better when the noise is added to the basic sensory inputs, because its effects are reduced by the preprocessing applied to obtain the input signals.

Figure 9, panel B, illustrates some of the generalization abilities of the path-finder. It can face many more situations than those perceived during the learning phase. In addition, the path-finder is also able to navigate in workspaces different from the one used during learning. Figures 11 through 13 show the behavior of the path-finder when only the goal is changed, more obstacles are added to the original workspace, and both the goal and

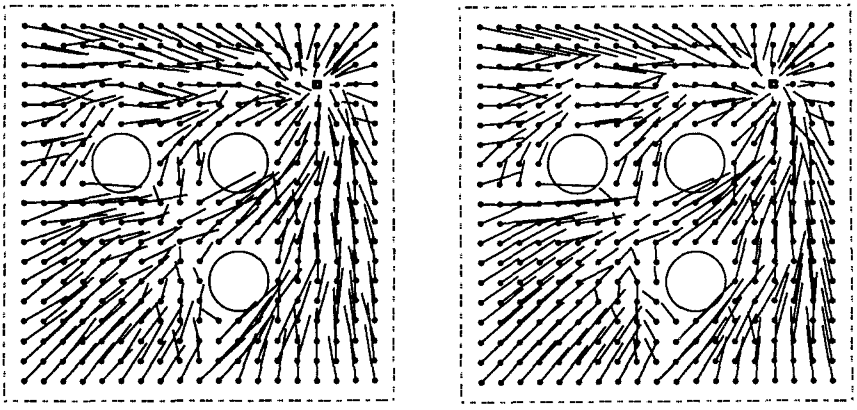

(a) (b)

Figure 10. Noise tolerance exhibited by the path-finder.

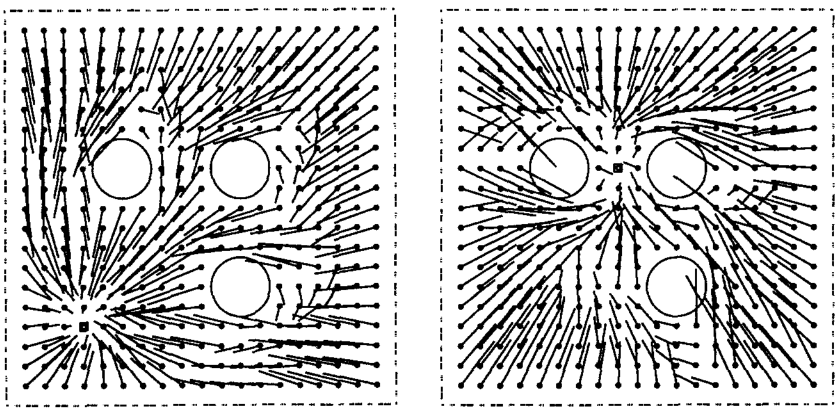

(a) (b)

Figure 11. Generalization abilities: New goals.

the number of obstacles have changed. The results of these three experiments show that the *situation-action rules learned are both goal-independent and workspace-independent* and that, even in the worst cases, only a "light" additional learning phase suffices to readapt the path-finder to the new workspace.

Finally, Figure 14 shows how the *path-finder copes with dynamic environments*. If the robot has taken one or more steps toward the goal and either the obstacles (panel A) or the goal (panel B) move, the path-finder is still able to generate feasible paths. In the first case, the obstacles are moving toward the northwest, therefore approaching the goal, and

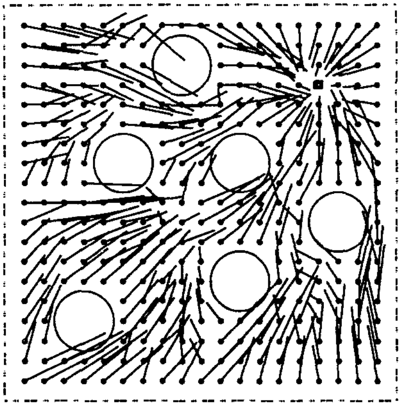


Figure 12. Generalization abilities: More obstacles.

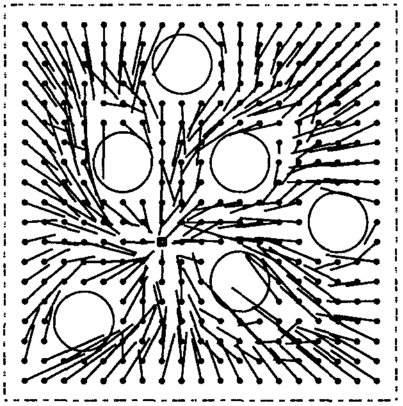


Figure 13. Generalization abilities: New goal and more obstacles.

the path-finder avoids them. In the second case, the goal is moving toward the northeast and the path-finder tracks it. This ability could be enhanced if a module to predict the motion of the goal and the obstacles were incorporated into the system.

6. Conclusions and future work

The simulations carried out in this paper demonstrate the adequacy of a reinforcement connectionist learning approach to implement local obstacle-avoidance capabilities. The formulation of the problem used to test this approach is a difficult one, since the input and output are continuous, the environment is partially unknown, and the optimality criterion (finding short paths with wide clearances) is severe. The problem, however, has been simplified by assuming a point robot and circular obstacles. These simplifications are dropped in the

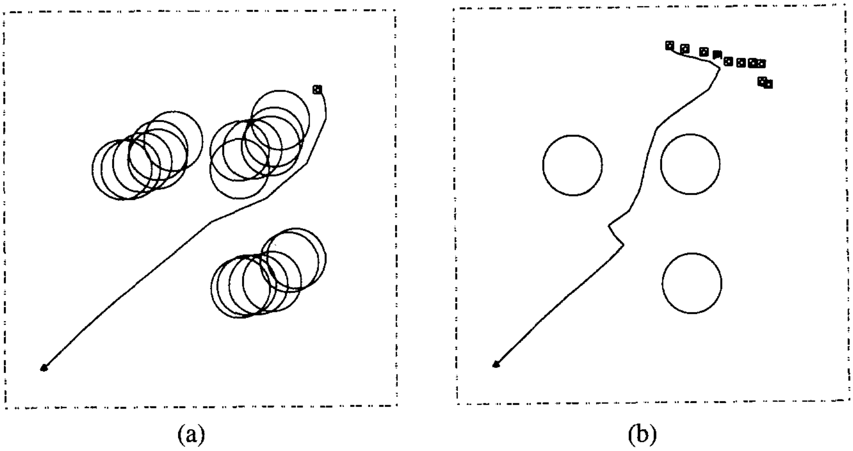


Figure 14. (a) Dynamic obstacles. (b) Dynamic goal.

extended prototype we are currently working on, which deals with a 2D mobile robot and polygonal obstacles.

The codification scheme adopted and the algorithm used to discover qo-paths can be thought of as domain-specific heuristics (Langley, 1985) for the robot path finding problem, which greatly improve the basic reinforcement-based weak search method. Equipped with these two modules, the path-finder not only learns the necessary situation-action rules in a very reduced time, but also exhibits good noise-tolerance and generalization capabilities, and is able to cope with dynamic environments.

In the Introduction we claimed that the robot path finding problem could be solved efficiently if symbolic planning were interfaced to subsymbolic obstacle avoidance. Our current research is oriented towards designing a *hybrid path-finder* by coupling a geometric global planning approach such as that in Ilari and Torras (1990) with the connectionist local obstacle avoidance approach described in this paper.

In Millán and Torras (1991b) we illustrate with a very simple example the potential benefits of integrating symbolic and subsymbolic techniques according to this general framework. A symbolic path-planner suggests intermediate configurations—subgoals or landmarks—that the path has to go through. The path-planner is invoked both at the beginning of the task and whenever the course of action seems to be wrong. This happens when the path-finder cannot find suitable action to handle the current situation or when the robot deviates considerably from the planned physical path.

Acknowledgments

We thank Rich Sutton whose suggestions have been really valuable for improving the presentation of our work, and Aristide Varfis for helpful discussions. We would like also to express our gratitude to two of the reviewers for their detailed comments. Carme Torras has been partially supported by the ESPRIT Basic Research Action number 3234.

Notes

1. Although the notion of *configuration space* is that of a "state space" and thus it is quite old, the term itself was first introduced by Lozano-Pérez and Wesley (1979) and subsequently developed in Lozano-Pérez (1983). Roughly speaking, it is the *space of degrees of freedom of motion* in which a given motion planning problem is to be solved.
2. If the reinforcement signal were computed using *local information*, then the task would be to select actions that optimize the *cumulative reinforcement* received by the path-finder over time. A prediction problem would have to be solved, *temporal-difference methods* (Sutton, 1988; Barto, et al., 1989; Watkins, 1989) having proved to be useful for this kind of task.

References

Agre, P.E., & Chapman, D. (1987). Pengi: An implementation of a theory of activity. *Proceedings of the Seventh AAAI Conference* (pp. 268-272).

Anderson, C.W. (1986). *Learning and problem solving with multilayer connectionist systems.* Ph.D. Thesis, Dept. of Computer and Information Science, University of Massachusetts, Amherst.

Anderson, C.W. (1987). Strategy learning with multilayer connectionist representations. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 103-114).

Arkins, R.C. (1987). Motor schema based navigation for a mobile robot: An approach to programming by behavior. *Proceedings of the IEEE International Conference on Robotics and Automation* (pp. 264-271).

Barto, A.G. (1985). Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology, 4,* 229-256.

Barto, A.G., & Anandan, P. (1985). Pattern-recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics, 15,* 360-374.

Barto, A.G., Sutton, R.S., & Anderson, C.W. (1983). Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics, 13,* 835-846.

Barto, A.G., Sutton, R.S., & Brouwer, P.S. (1981). Associative search network: A reinforcement learning associative memory. *Biological Cybernetics, 40,* 201-211.

Barto, A.G., Sutton, R.S. & Watkins, C.J.C.H. (1989). *Learning and sequential decision making* (Technical Report COINS-89-95). University of Massachusetts, Amherst, MA: Dept. of Computer and Information Science.

Blythe, J., & Mitchell, T.M. (1989). On becoming reactive. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 255-259).

Brady, M., Hollerbach, J.M., Johnson, T.L., Lozano-Pérez, T., & Mason, M.T., (Eds.) (1982). *Robot motion: Planning and control.* Cambridge, MA: MIT Press.

Brooks, R.A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation, 2,* 14-23.

Canny, J.F. (1988). *The complexiity of robot motion planning.* Cambridge, MA: MIT Press.

Chapman, D., & Kaelbling, L.P. (1990). *Learning from delayed reinforcement in a complex domain* (Technical Report 90-11). Palo Alto, CA: Teleos Research.

Donald, B.R. (1987). A search algorithm for robot motion planning with six degrees of freedom. *Artificial Intelligence, 31,* 295-353.

Graf, D.H., & LaLonde, W.R. (1988). A neural controller for collision-free movement of general robot manipulators. *Proceedings of the IEEE Second International Conference on Neural Networks, Vol I* (pp. 77-84).

Gullapalli, V. (1988). *A stochastic algorithm for learning real-valued functions via reinforcement feedback* (Technical Report COINS-88-91). University of Massachusetts, Amherst, MA: Dept. of Computer and Information Science.

Ilari, J., & Torras, C. (1990). 2D path planning: A configuration space heuristic approach. *The International Journal of Robotics Research, 9,* 75-91.

Jordan, M.I., & Jacobs, R.A. (1990). Learning to control an unstable system with forward modeling. In D.S. Touretzky (Ed.), *Advances in neural information procesing systems 2,* 324-331. San Mateo, CA: Morgan Kaufmann.

Jorgensen, C.C. (1987). Neural network representation of sensor graphs in autonomous robot navigation. *Proceedings of the IEEE First International Conference on Neural Networks, Vol IV* (pp. 507–515).

Khatib, O. (1986). Real time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research, 5,* 90–98.

Langley, P. (1985). Learning to search: From weak methods to domain-specific heuristics. *Cognitive Science, 9,* 217–260.

Lin, L.-J. (1990). Self-improving reactive agents: Case studies of reinforcement learning frameworks. *Proceedings of the First International Conference on the Simulation of Adaptive Behavior: From Animals to Animats* (pp. 297–305).

Lozano-Pérez, T. (1983). Spatial planning: A configuration space approach. *IEEE Transactions on Computers, 32,* 108–120.

Lozano-Pérez, T., & Wesley, M. (1979). An algorithm for planning collison-free paths among polyhedral obstacles. *Communications of the ACM, 22,* 560–570.

Mahadevan, S., & Connell, J. (1990). *Automatic programming of behavior-based robots using reinforcement learning* (Technical Report RC 16359). Yorktown Heights, NY: IBM, T.J. Watson Research Center.

Mel, B.W. (1989). MURPHY: *A neurally-inspired connectionist approach to learning and performance in vision-based robot motion planning.* Ph.D. Thesis, Graduate College, University of Illinois, Urbana-Champaign.

Millán, J. del R., & Torras, C. (1990). Reinforcement learning: Discovering stable solutions in the robot path finding domain. *Proceedings of the Ninth European Conference on Artificial Intelligence* (pp. 219–221).

Millán, J. del R., & Torras, C. (1991a). Connectionist approaches to robot path finding. In O.M. Omidvar (Ed.), *Progress in neural networks series, Vol 3.* Norwood, NJ: Ablex.

Millán, J. del R., & Torras, C. (1991b). Learning to avoid obstacles through reinforcement. In L. Birnbaum & G. Collins (Eds.) *Machine learning: Proceedings of the Eighth International Workshop,* 298–302. San Mateo, CA: Morgan Kaufmann.

Mozer, M.C., & Bachrach, J. (1989). *Discovering the structure of a reactive environment by exploration* (Technical Report CU-CS-451-89). Boulder, CO: University of Colorado, Dept. of Computer Science.

Munro, P. (1987). A dual back-propagation scheme for scalar reward learning. *Proceedings of the Ninth Annual Conference of the Cognitive Science Society* (pp. 165–176).

Rivest, R.L., & Schapire, R.E. (1987). A new approach to unsupervised learning in deterministic environments. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 364–375).

Robinson, A.J. (1989). *Dynamic error propagation networks.* Ph.D. Thesis, Engineering Department, Cambridge University, Cambridge, England.

Saerens, M., & Soquet, A. (1989). A neural controller. *Proceedings of the First IEE International Conference on Artificial Neural Networks* (pp. 211–215).

Schoppers, M.J. (1987). Universal plans for reactive robots in unpredictable environments. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 1039–1046).

Singh, S.P. (1991). Transfer of learning across compositons of sequential tasks. In L. Birnbaum & G. Collins (Eds.) *Machine learning: Proceedings of the Eighth International Workshop,* 348–352. San Mateo, CA: Morgan Kaufmann.

Steels, L. (1988). Steps towards common sense. *Proceedings of the Eighth European Conference on Artificial Intelligence* (pp. 49–54).

Sutton, R.S. (1984). *Temporal credit assignment in reinforcement learning.* Ph.D. Thesis, Dept. of Computer and Information Science, University of Massachusetts, Amherst.

Sutton, R.S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning, 3,* 9–44.

Sutton, R.S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 216–224).

Torras, C. (1990). Motion planning and control: Symbolic and neural levels of computation. *Proceedings of the Third COGNITIVA Conference* (pp. 207–218).

Watkins, C.J.C.H. (1989). *Learning with delayed rewards.* Ph.D. Thesis, Psychology Department, Cambridge University, Cambridge, England.

Werbos, P.J. (1987). Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research. *IEEE Transactions on Systems, Man, and Cybernetics, 17,* 7–20.

Whitesides, S.H. (1985). Computational geometry and motion planning. In G. Toussaint (Ed.), *Computational geometry.* Amsterdam, New York, Oxford: North-Holland.

Williams, R.J. (1986). *Reinforcement learning in connectionist networks: A mathematical analysis* (Technical Report ICS-8605). San Diego, CA: University of California, Institute for Cognitive Science.

Williams, R.J. (1987). *Reinforcement-learning connectionist systems* (Technical Report NU-CCS-87-3). Northeastern University, Boston, MA: College of Computer Science.

Yap, C.-K. (1987). Algorithmic motion planning. In J.T. Schwartz & C.-K. Yap (Eds.), *Advances in robotics, Vol. I. Algorithmic and geometric aspects of robotics*, 95–143. Hillsdale, NJ: Lawrence Erlbaum.

Appendix A: Computation of the repulsion forces

The intensity of each repulsion force depends on three factors, namely r_a, r_b and r_c. Assuming circular obstacles, each obstacle is determined by three parameters: the *coordinates* of the *center* of the circle (x_o, y_o) and the *radius* ra. If the current and goal configurations are the points (x_a, y_a) and (x_g, y_g), respectively, then the points of the SPV are defined by the equation:

$$(x_{SPV}, y_{SPV}) = (1 - p)(x_a, y_a) + p(x_g, y_g), \tag{28}$$

where $p \in [0, 1]$. So, each point of conflict (x_c, y_c) is determined by the value p_c which solves the differential equation:

$$\frac{d \operatorname{dist}[(x_o, y_o), (x_c, y_c)]}{dp} = 0, \tag{29}$$

and by the additional constraint that it must be outside the corresponding obstacle. This value is:

$$p_c = \operatorname{box}_1 \left(- \frac{(x_o - x_a)(x_a - x_g) + (y_o - y_a)(y_a - y_g)}{(x_a - x_g)^2 + (y_a - y_g)^2} \right), \tag{30}$$

where

$$\operatorname{box}_1(x) = \begin{cases} 1, & \text{if } x > 1, \\ 0, & \text{if } x < 0, \\ x, & \text{otherwise.} \end{cases} \tag{31}$$

Now, the three repulsion factors are given by the expressions:

$$r_a = \operatorname{box}_2(\operatorname{dist}[(x_o, y_o), (x_c, y_c)], ra), \tag{32}$$

$$r_b = \operatorname{box}_3(\operatorname{dist}[(x_a, y_a), (x_c, y_c)], \operatorname{dist}[(x_o, y_o), (x_c, y_c)], ra), \tag{33}$$

$$r_c = \operatorname{box}_4(\operatorname{dist}[(x_o, y_o), (x_c, y_c)], ra), \tag{34}$$

where

$$\mathrm{box}_2(x, y) = \begin{cases} 0, & \text{if } x < y, \\ limit(x - y), & \text{otherwise,} \end{cases} \tag{35}$$

$$\mathrm{box}_3(x, y, z) = \begin{cases} limit(x), & \text{if } y > z, \\ limit(x - \sqrt{z^2 - y^2}), & \text{otherwise,} \end{cases} \tag{36}$$

$$\mathrm{box}_4(x, y) = \begin{cases} 0, & \text{if } x > y, \\ y - x, & \text{otherwise,} \end{cases} \tag{37}$$

and

$$limit(x) = \begin{cases} k_{pa} * x, & \text{if } x > perception_{range}, \\ x, & \text{otherwise.} \end{cases} \tag{38}$$

This *limit* function is defined in such a way as to keep the repulsion forces very small in those cases where obstacles are located outside the *perception range* of the robot (see (39) below). k_{pa} and $perception_{range}$ are constants, $k_{pa} \gg 1$, and $perception_{range}$ is chosen to be a fifth of the dimension of the workspace

Once r_a, r_b and r_c have been calculated, the intensity of a repulsion force is:

$$ir(r_a, r_b, r_c) = e^{-(k_{r_a} * r_a + k_{r_b} * r_b)} + \frac{1}{1 - e^{k_{r_c} * r_c}} - \frac{1}{2}, \tag{39}$$

where k_{r_a}, k_{r_b} and k_{r_c} are constants.

The intensity of the environmental repulsion force from a quadrant is:

$$r^i(x) = \begin{cases} 1, & \text{if a collision happens in the quadrant } i, \\ \dfrac{1}{1 + e^{-(k_{rep} * x)}}, & \text{otherwise,} \end{cases} \tag{40}$$

where k_{rep} is a constant and x is the sum of the intensities of the repulsion forces from quadrant i.

Appendix B: Analysis of variance

Table 6 provides the standard deviation in the number of steps for finding a stable qo-path of the most promising versions. Table 7 shows the *significance degree of the means differences* when every version is compared with each other.

Table 6. Standard deviation in the number of steps for finding a stable qo-path.

		Eligibility		
		3	4	5
	1	15654	11477	5249
Baseline	4	9541	9814	8655
	5	9145	9318	10948

Table 7. Significance degrees of the means differences. $\mu_1 < \mu_2$, where μ_i is the mean of the sample i.

	Sample 2								
Sample 1	v_{45}	v_{15}	v_{54}	v_{43}	v_{44}	v_{14}	v_{13}	v_{53}	v_{55}
v_{45}		80.23%	87.08%	92.07%	96.25%	96.64%	95.73%	99.65%	99.84%
v_{15}			70.54%	81.06%	90.66%	91.92%	90.99%	99.20%	99.62%
v_{54}				61.03%	74.54%	78.81%	81.59%	93.32%	96.78%
v_{43}					64.43%	70.54%	75.49%	88.49%	94.18%
v_{44}						57.53%	65.54%	78.81%	88.49%
v_{14}							59.10%	69.85%	82.12%
v_{13}								56.75%	70.19%
v_{53}									68.79%
v_{55}									

In order to compute the significance degree of the difference between a pair of sample means μ_1 and μ_2, the following *discriminant function* is built:

$$u = (\mu_1 - \mu_2)\sqrt{\frac{1}{\dfrac{\sigma_1^2}{n1 - 1} + \dfrac{\sigma_2^2}{n2 - 1}}}, \tag{41}$$

where $n1$ and $n2$ are the size of each sample, and σ_1 and σ_2 are the sample standard deviations. Now, if experimentally $\mu_2 > \mu_1$, then for estimating the significance degree $1 - e$ it is necessary to find u_{1-e}, in the table of the normal distribution, such that $u < -u_{1-e}$.

Data in Table 7 confirm the results of Section 5.1. In addition, it is possible to draw the following two main conclusions. First, the versions are ranked in the following order: v_{45}—i.e., the version using the fourth baseline an fifth eligibility factor—v_{15}, v_{54}, v_{43}, v_{44}, v_{14}, v_{13}, v_{53} and v_{55}. Second, v_{45} is *significantly better* then v_{53} and v_{55}—the corresponding significance degrees are greater than 99%—it is *quasi-significantly better* than v_{13}, v_{14} and v_{44}—the corresponding significance degrees are greater than 95%—it is *presumably better* than v_{43}—the significance degree is greater than 90%—and it is *likely better* than v_{15} and v_{54}—the corresponding significance degrees are greater than 80%.