Improving reinforcement learning through a better exploration strategy and an adjustable representation of the environment

R. Iglesias* M. Rodríguez* M. Sánchez* E. Pereira* C.V. Regueiro†
* *Electronics and Computer Science, University of Santiago de Compostela, Spain.*
†*Electronic and Systems, University of Coruña, Spain*

***Abstract*— Reinforcement learning is a promising strategy as all the robot needs to start a random search of the desired solution is a reinforcement function which specifies the main restrictions of the behaviour. Nevertheless, the robot wastes too much time trying the execution of random –mostly wrong– actions, and the user is forced to determine the balance between the exploration of new actions and the execution of already tried ones. In this context we propose a methodology which is able to achieve fast convergences towards good robot-control policies, and it determines on its own the required degree of exploration at every instant. The performance of our approach is due to the mutual and dynamic influence that three different elements exert on each other: *reinforcement learning*, *genetic algorithms*, and a *dynamic representation of the environment around the robot*.**

In this paper we describe the application of our approach to solve two common tasks in mobile robotics (wall following and door traversal). The experimental results show how the required learning time is significantly reduced and the stability of the process is increased. On the other hand, the low user-intervention required to solve both tasks –only the reinforcement function is changed–, confirms the contribution of this approach towards robot techniques that are fast, user friendly, and demand little application-specific knowledge by the user, something more and more required nowadays.

***Index Terms*— Reinforcement learning, robot control, autonomous agents, genetic algorithms.**

I. Introduction

Reinforcement learning is a very interesting strategy, since all the robot needs for learning a behaviour is a reinforcement function which tells the robot how good or bad it has performed, but nothing about the set of actions it should have carried out. Through a stochastical exploration of the environment, the robot must find a control policy which maximizes the expected total reinforcement it will receive; Eq. 1:

$$E[\sum_{t=0}^{\infty}\gamma^t r_t] \tag{1}$$

where r_t is the reinforcement received at time *t*, and $\gamma \in [0,1]$ is a discount factor which adjusts the relative significance of long-term rewards versus short-term ones. Q-learning [9] is one of the most popular RL algorithms, through which the robot will learn a utility function of states and actions, termed Q-function. This function estimates the expected discounted reward for every state-action pair; i.e. Q(s,a) is the expected sum of future rewards obtained by taking action a in state s and following an optimal policy thereafter.

$$Q(s,a) = E[r(s,a) + \sum_{t=1}^{\infty}\gamma^t r_t | s_0 = s, a_0 = a]$$

r(s,a) is the reinforcement the robot receives when it executes action a in state s, and $E[\sum_{t=1}^{\infty}\gamma^t r_t | s_0 = s, a_0 = a]$ represents the expected total reinforcement the robot will receive starting from state *s*, by executing action *a*, and then following the best possible policy.

Once these Q-values have been learnt, the optimal action for any state is the one with the highest Q-value; this is called greedy policy:

$$\pi^*(s) = arg \quad max_a\{Q(s,a)\} \tag{2}$$

Q-learning is currently one of the strongest and most widely used reinforcement learning algorithms. Basically the robot begins with an initial set or random negative Q-values: $Q(s,a) \leq 0$, $\forall s, a$, and then it moves around interacting with the environment. At each time step the robot observes the current state of the environment, s_t, and selects an action a_t. After the execution of a_t in s_t, the robot receives a reinforcement value and it changes into state s_{t+1}. Starting from this information the system updates the estimation of how good or bad the execution of a_t in s_t seems to be:

$$\Delta Q(s_t, a_t) = \beta(r_t + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)), \tag{3}$$

where $\beta \in [0,1]$ is a learning rate.

Equation 3 describes the simplest version of Q-learning, as it only updates Q-values based on the transition between two consecutive states. When rewards occur infrequently, it can take many learning trials for these values to propagate to previous states. Due to this, what is termed TD-learning [7] attempts to expedite this process by simply adding more *memory* into the system. There are two well known options: eligibility traces [1], and truncated temporal differences – $TTD(\lambda, m)$ – [4, 3]. In the case of the $TTD(\lambda, m)$, to understand the way it works we need only to consider the example shown in Figure 1. According to this figure the last three states the robot visited are s_{t-3}, s_{t-2}, and s_{t-1}. In those states the robot performed actions a_{t-3}, a_{t-2} and a_{t-1}, and received r_{t-3}, r_{t-2} and r_{t-1} reinforcements. The robot's current state is s_t. If we wish to estimate the discounted

reinforcement the robot will receive after the execution of action a_{t-1} in s_{t-1} there is only one way that this can be done (Eq. 4):

$$Z_{t-1} = r_{t-1} + \gamma Qmax_a(s_t, a) \quad (4)$$

Nevertheless, when the state s_{t-2} is considered, there are two ways of proceeding:

$$source1 := r_{t-2} + \gamma Qmax_a(s_{t-1}, a) \quad (5)$$

$$source2 := r_{t-2} + \gamma r_{t-1} + \gamma^2 Qmax_a(s_t, a) = r_{t-2} + \gamma Z_{t-1} \quad (6)$$

Due to this, both sources are averaged to estimate the discounted reinforcement the robot will receive after execution action a_{t-2} in s_{t-2}:

$$Z_{t-2} = r_{t-2} + \gamma[\lambda Z_{t-1} + (1-\lambda)Qmax_a(s_{t-1}, a)].$$

where $\lambda \in [0, 1]$.

The same assumptions hold for s_{t-3}, Figure 1.

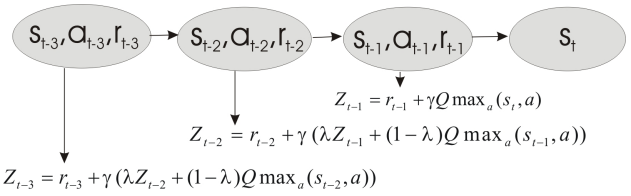

Fig. 1. Short Markov chain showing the last three states the robot went through. The expected future reinforcement the robot will receive is estimated for every state.

In general, through the $TTD(\lambda, m)$ algorithm, the robot keeps track of the last m-executed actions ($m = 3$ in the example shown in Figure 1), so that the Q-value corresponding to the pair s_{t-m}, a_{t-m} is updated according to the estimated discounted reward Z_{t-m}:

$$\Delta Q(s_{t-m}, a_{t-m}) = \beta(Z_{t-m} - Q_t(s_{t-m}, a_{t-m})),$$

where Z_{t-m} is obtained recursively as described before:

$$Z_{t-k} = r_{t-k} + \gamma[\lambda Z_{t-k+1} + (1-\lambda)Qmax_a(s_{t-k+1}, a)], \quad \forall k = 1, ..., m$$

Despite the benefits of the RL paradigm in autonomous robot-learning, there are important problems to consider when it is applied. First, the time required to learn a good greedy policy (Eq. 2) increases exponentially with the number of states and the number of actions that are possible to execute in each state. On the other hand, the robot wastes an enormous amount of time trying actions that are clearly inappropriate for the task but which are selected randomly during the learning process.

II. Our approach

To solve some of the aforementioned drawbacks, we propose a learning strategy which combines three elements: *reinforcement learning (RL)*, *a genetic algorithm (GA)*, and an *adjustable number of states representing the environment around the robot*. Basically, when our approach is applied the robot goes through three cyclical and clearly differentiated stages (Figure 2): a) *looking for a new starting position or convergence*. b) *exploration*, and c) *generation of a new population of solutions (chromosomes) for the next exploration stage*.

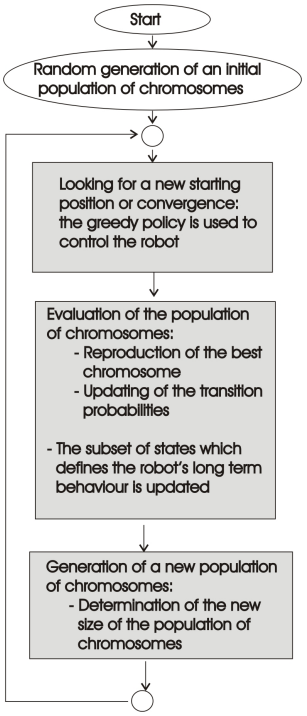

Fig. 2. Flow diagram which describes the combination of RL and GA.

During the first stage the robot uses the greedy policy to move according to what has been learnt so far. When the robot finds a new situation where it does not know how to move – local problem – (Figure 3.a), it applies a genetic algorithm to find a local solution (Figure 3.b). Once the solution has been achieved a new population of chromosomes is generated and the robot moves again using the greedy policy until a new local problem is detected (Figure 3.c).

According to this, our proposal looks very simple and it seems to be just the sequential combination of two different learning paradigms (RL and GA). Nevertheless this assumption would be erroneous, as it is the *mutual influence* between the three elements aforementioned (RL, GA, and adjustable number of states), what really improves the learning procedure. In fact, this mutual influence varies throughout the learning process: the influence of the GA is very high at the beginning but it decreases over time. The influence of the RL is higher and higher as the number of robot-environment interactions increases and, finally, the dynamic number of states helps the robot to learn first how to behave in those situations that are most frequent, while the less frequent ones are learnt in a second stage.

A. Looking for a new starting position

During this first stage the greedy policy – which represents what the robot has learnt so far– is applied to control the robot. If the robot encounters a situation where it does not know how

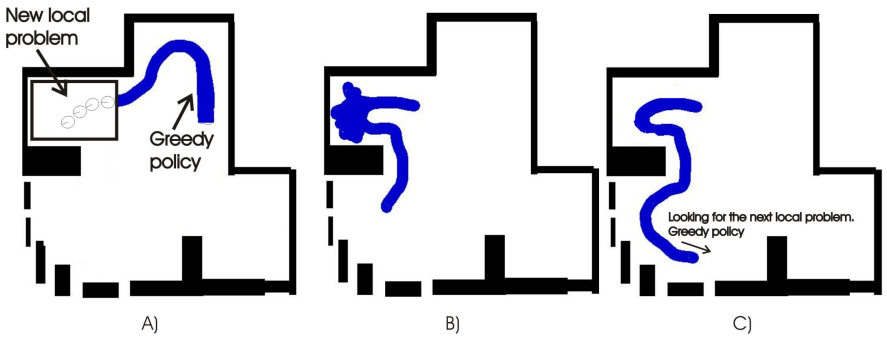

Fig. 3. Schematic representation of our approach. Initially the robot moves using the greedy control policy until it finds a situation it doesn't know how to solve (a), a genetic algorithm is applied to find a solution (b), once the problem is solved the greedy policy is applied again (c).

to move – local problem, (Figure 3.a) –, it receives a sequence of consecutive negative reinforcements, and its position several steps before the failure is established as a new starting position for the next exploration stage.

B. Exploration stage

Our strategy applies a GA in order to solve each local problem the robot finds. The GA starts with a population of solutions called chromosomes. Each chromosome (represented as π) determines the action the robot has to carry out at each state, s: $\pi(s)$. The population of chromosomes is evaluated according to an objective function called *fitness* function, which reflects for how long a chromosome is able to properly control the movement of the robot (Figure 3.b). Once a population of chromosomes has been evaluated, the sequence of states, actions, and rewards the robot received under the control of the best chromosome, is replicated off-line several times to speed up the convergence of the Q-values.

C. Generation of a new population of solutions (chromosomes)

The population of chromosomes has to be evolved according to the fitness values. In order to do this, certain genetic operators like mutation –which carries out random changes in the chromosomes–, or chromosome crossover –which combines chromosomes to raise new solutions– have to be applied. We use the Q-values to bias the genetic operators and thus reduce the number of chromosomes which are required to find a solution. Given a particular chromosome π, the probability that mutation changes the action that this chromosome suggests for a particular state s: $\pi(s)$, depends on how many actions appear better or worse than $\pi(s)$ according to the Q-values:

$$P_{mutation}(\pi(s)) = \frac{N_1}{N - N_1}$$

where N is the number of actions and N_1 represents the number of actions that appear better than $\pi(s)$:

$$N_1 = cardinality\{a_j \in A \mid Q(s,a_j) \geq Q(s,\pi(s))\}$$

Finally, if the action that chromosome π suggests for state s is to be changed, we use another probability which also considers the Q-values in order to determine a new action to replace $\pi(s)$:

$$P_s(s,a_i) = \frac{e^{Q(s,\pi(s))/Q(s,a_i)}}{\sum_j e^{Q(s,\pi(s))/Q(s,a_j)}}. \quad (7)$$

One of the chromosomes should always be the greedy policy as it brings together all that has been already learnt by the robot, and it represents the best chance of obtaining a fast convergence towards the desired solution.

Finally, when the robot is looking for a new starting position and the greedy policy is being used to control it, if the robot moves properly during *M* steps before it receives negative reinforcements, only the states involved in the robot's last *K* movements are susceptible of being changed through the GA, while the states involved in the initial *M-K* actions are labelled as learnt, so that neither chromosome selection nor chromosome crossover can alter them.

The population of chromosomes is resized after its evaluation according to how close the GA is to the desired solution.

D. Adjustable number of states

The number of states used to represent the environment around the robot influences the time required to learn a satisfactory greedy policy. For this reason we use the properties of the regular Markov chains [2] to dynamically adjust the number of states involved in the learning process.

As the robot is able to translate the different situations it may detect through its sensors into a finite number of states, $S = \{s_1, .., s_N\}$, there is a $N \times N$ transition matrix, P, so that if the robot is in state s_i at time step t-1, there is a fixed probability, P_{ij}, of it going into state s_j:

$$P_{ij} = Prob\{s_t = j | s_{t-1} = i\}$$

Therefore, if we have some information about the state where the robot is at time instant t, we can try to predict where the robot will be at $t+1$, Eq. 8:

$$\overrightarrow{\chi_{t+1}} = (\chi_{t+1}(s_0), \chi_{t+1}(s_1), ..., \chi_{t+1}(s_N)) = \overrightarrow{\chi_{\mathbf{t}}} P \quad (8)$$

$\chi_{t+1}(s_i)$ represents the probability that in the time instant t+1 the robot is in state s_i, $\forall i = 1, ..., N$. $\overrightarrow{\chi_{\mathbf{t}}}$ represents the probability distribution corresponding to the time instant t.

If P is the transition matrix for a Markov system, and $\overrightarrow{\chi_{t+k}}$ is a distribution vector with the property that $\overrightarrow{\chi_{\mathbf{t+k+1}}} =$

$\overrightarrow{\chi_{t+1}} P^k = \overrightarrow{\chi_{t+k}}$, then we refer to $\overrightarrow{\chi_{t+k}}$ as a *steady vector*. This means that the probability of finding the robot in state *i* is the same all the time: although the robot is jumping from state to state, the probability distribution looks the same. As the steady vector $\overrightarrow{\chi_{t+k}}$ does not depend on t, we will dispense with the subscript and denote it merely as $\overrightarrow{\chi}$. When the transition matrix P satisfies certain conditions (some power of P has no zero entries), it is proved that $\overrightarrow{\chi}$ exists and it is unique.

In our case, during the learning process the transition matrix is estimated so that $\overrightarrow{\chi}$ can be calculated. Once this is done, and since $\overrightarrow{\chi}$ defines how the robot sees the environment in the long-term, only those states s_j for which the long term probability is not null ($\chi(s_j) \neq 0$), are considered in the learning procedure.

To finish section II, we will mention some of the differences between our approach and a *Learning Classifier System*, specifically the XCS. While XCS is a rule-based learning strategy which relies on the combination of GA and RL to build the rule set that it manipulates, our proposal doesn't use the combination GA-RL to build a dynamic representation of the environment but to get a fast reduction of the degree of randomness in the actions that are executed. Finally, the combination of GA and RL is done through the fitness function in the XCS strategy, while our proposal locates the combination on the genetic operators – the way mutation, selection and crossover work depend on the Q-values learnt through the RL algorithm–.

III. Experimental results

We applied our approach to solve two common tasks: "wall following" and "door traversal". We used a Nomad200 robot equipped with 16 ultrasound sensors encircling its upper part and bumpers. In all the experiments the linear velocity of the robot was kept constant (15.24 cm/s), and the robot received the commands it had to execute every 300ms.

We used a set of two-layered Kohonen networks to translate the large number of different situations that the ultrasound sensors located on the front and right side of the robot may detect, into a finite set of 220 neurones – states – [6].

The robot was taught how to perform each task in a simulated training environment, but its performance was tested in a different one. Convergence was detected when the greedy policy was able to properly control the movement of the robot for an interval of 10 minutes.

Through these experimental results we want to put emphasis on the following aspects: a) the performance of our approach is higher than the classical RL algorithms, not only under normal working conditions but also when the parameters of the classical RL algorithm (β,γ,m,λ) are optimized (section A). b) The learning process we get through our proposal is stable (section B). c) The results we got are valid even when the task to be learnt is changed (section C).

A. Wall following

To teach the robot how to follow a wall located on its right at a certain distance interval, we used a reinforcement signal that is negative whenever the robot goes too far from or too close to the wall being followed.

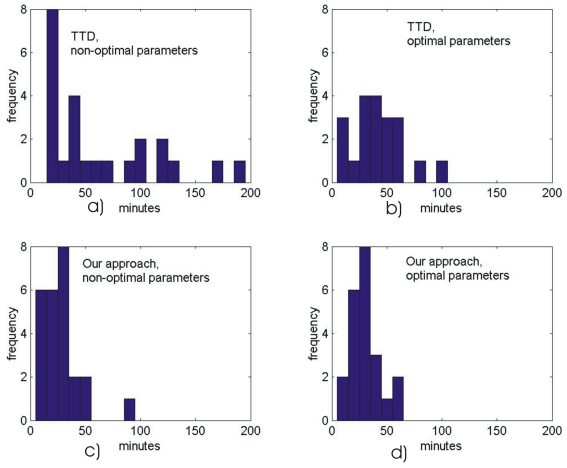

Fig. 4. Distribution of the learning times obtained after running 24 experiments using $TTD(\lambda, m)$ with a set of parameters manually selected (a), 20 experiments using $TTD(\lambda, m)$ with an optimal combination of the parameters β, γ, λ, and m (b), 25 experiments using our approach with the same manual selection of parameters as before (c), and 22 experiments running our approach with the optimal combination of parameters as before (d).

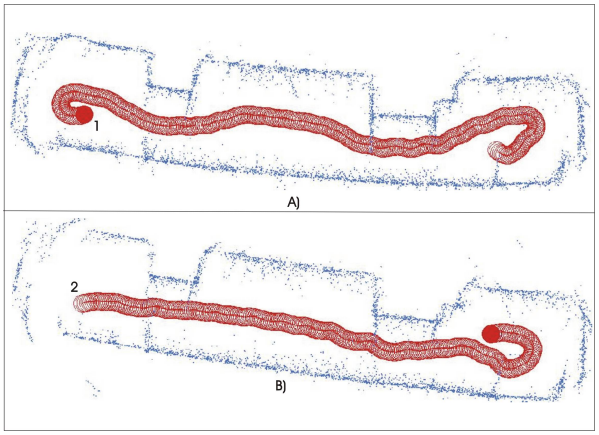

Fig. 5. Real robot's trajectory along a corridor when a control policy learnt through our approach was used. For a clear view of the trajectory, figure a) shows the robot's movement in one direction and b) shows the movement along the opposite direction. Points 1 and 2 in both graphs correspond to the same robot position. The small dots represent the ultrasound readings.

We first decided to apply the $TTD(\lambda, m)$ when the parameters β, λ, m and γ (see section I), were selected following two different strategies: (a) manual selection, and (b) automatic search for the optimal combination. In the former the values of the parameters were selected as the optimal ones after trying with a small set of different combinations: $\beta = \{0.25, 0.35, 0.45, 0.55\}$, $\lambda = \{0.4, 0.5, 0.60.7, 0.8\}$, and $m = \{10, 20, 30, 40, 50\}$, the selected combination was $\beta = 0.35$, $\lambda = 0.8$, $\gamma = 0.95$ and $m = 30$.

Secondly, we applied a GA to search for a combination of parameters through which the learning time is optimized. Thus, after running the GA for a minimum period of 30 hours

TABLE I

AVERAGE LEARNING TIME AND STANDARD DEVIATIONS (σ) OBTAINED WHEN THE PARAMETERS λ, β, γ AND m WERE OPTIMIZED THROUGH A GA, AND WHEN THEY WERE MANUALLY SELECTED.

Strategy	average learning time (minutes)	σ (minutes)
$TTD(\lambda, m)$ manual selection of β, γ, λ, m	64.42 24 experiments	49.82
$TTD(\lambda, m)$ optimal combination of β, γ, λ, m	41.27 20 experiments	22.5
Our approach manual selection of β, γ, λ, m	28.80 25 experiments	16.62
Our approach optimal combination of β, γ, λ, m	30.83 22 experiments	12.64

on a Pentium M 1.60Ghz, the best combination we found was: $\beta = 0.288282$, $\lambda = 0.869965$, $\gamma = 0.796462$ and $m = 53$. To avoid falling into the habitual " conspiracy of goodwill" trap [5], different persons carried out each one of the parameter selection strategies.

The experimental results shown in Table I and Figures 4.a and 4.b reflect the advantages of the optimal combination of the parameters in comparison with the manually selected ones. Through the results we can see how the learning time and the standard deviation are significantly reduced when the $TTD(\lambda, m)$ algorithm is executed with the parameters determined through the GA.

When we applied our approach, the average learning time and the standard deviation were much lower than with the $TTD(\lambda, m)$ algorithm (Table I, Figures 4.c and 4.d). According to the results, the sensitivity with respect to the parameters λ, β, γ and m is almost inexistent, as the average learning times are roughly the same: 28.80 minutes (manual selection of parameters) and 30.83 minutes (optimal combination of parameters)– the 2 minutes difference represents nothing if we consider the standard deviations. This result is interesting, since it means that a reasonable selection of the parameters is sufficient to obtain a fast robot-learning of the desired behaviour. In all the experiments with our approach the number of chromosomes varied dynamically within the interval $[3, 20]$.

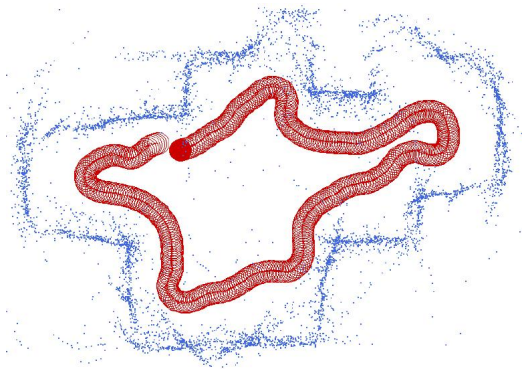

Fig. 6. Real robot's trajectory when the same control policy as in figure 5 was used.

To prove that the behaviours learnt through our approach are useful, Figures 5 and 6 show the movement of the robot in two real and noisy environments.

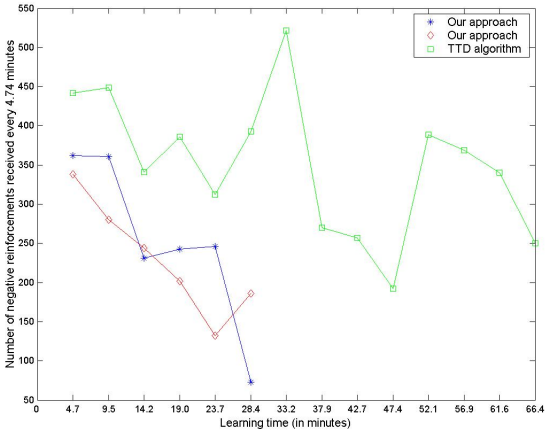

Fig. 7. Number of negative reinforcements the robot receives every 4.7 minutes during the learning stage. One of the curves represents one of our experiments using the $TTD(\lambda, m)$ algorithm, while the other two represent two experiments we carried out using our approach. All the experiments were randomly selected between those we carried out.

B. Stability of the learning procedure

We analysed the stability in two different ways: first of all, what we mean by stability is that the degree of randomness during the learning process is lower than with other classical RL algorithms. Figure 7 shows how through our proposal the number of wrong actions the robot tries during the learning process – reflected by the number of negative reinforcements it receives– decreased rapidly over time.

On the other hand, by stability we also mean Liapunov stability; i.e. small changes in the initial conditions should not give rise to very different final systems. Although the analysis of the Liapunov stability is still part of the ongoing research at the University of Santiago, work to date has consisted of the comparison of the steady vectors (Section II.D) corresponding to the different solutions achieved with both $TTD(\lambda, m)$, and our approach. Thus, if we consider the set of the steady vectors corresponding to the $TTD(\lambda, m)$ solutions, $SET\{\chi_{RL}\}$, and the corresponding average value $\overline{\chi_{RL}}$, a mathematical analysis showed that the entire set $SET\{\chi_{RL}\}$ was inside a sphere centred in $\overline{\chi_{RL}}$, with radius 0.37. In the case of the steady vectors corresponding to the solutions found through our approach $SET\{\chi_{OA}\}$, and its average value $\overline{\chi_{OA}}$, we could see that the entire set was inside a sphere centred in $\overline{\chi_{OA}}$ and radius 0.02. It is important to notice that these radii are in probability units – the set of all possible solutions is inside a sphere of radius 1–.

C. Door traversal

We also applied our approach to solve the door traversal behaviour in the experimental scenario shown in Figure 8. To learn this task the reinforcement is negative whenever the robot collides with the door-frame, the robot goes too far from the door, or the movement direction is so unsuitable that the robot is not facing the door any more.

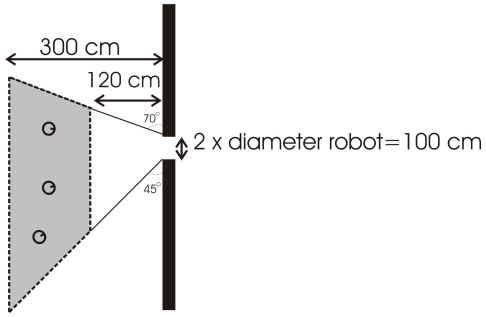

Fig. 8. Experimental scenario for the door traversal behaviour. The initial positions of the robot were within the shaded area.

After having carried out 21 experiments, the resulting average learning time was 86 minutes and 42 seconds – our approach was used with the same NON optimal combination of parameters as for the wall following behaviour, $\beta = 0.35$, $\lambda = 0.8$, $\gamma = 0.95$ and $m = 30$. Figure 9 shows four trajectories of the real robot across the door using the robot-controller learnt in simulation. There are several issues that should be remarked upon here: First, to solve the door traversal behaviour we are only using the ultrasound sensor readings (which proved to be very noisy for this task); the use of a more precise sensor, such as a laser scanner, would make the learning of this task easier. Second, the real scenario in which the robot is moved is not identical to the experimental scenario where the robot-controller was learnt (Figure 8), nevertheless the controller was robust enough to deal with these environment changes. Finally, the robot identifies the environment through the same set of states as those used in the wall following behaviour: two-layer Kohonen networks. Although these networks were trained to recognise the most frequent situations in a wall following behaviour and they only consider the ultrasound sensor readings coming from the right side of the robot, the analysis of the door traversal behaviour published in [8], proves that this information should be enough to solve the door-traversal task.

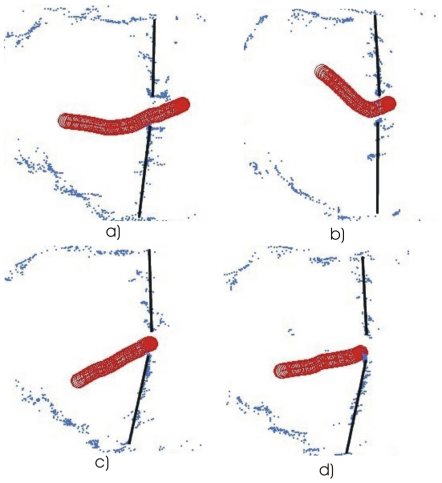

Fig. 9. Trajectories of the real robot across a door. The size of the door is 100 cm in a) and b), 90 cm in c), and 80 cm in d). The small dots represent the ultrasound readings.

IV. Conclusion

Through RL the robot is able to learn on its own – through trial an error interactions with the environment– using only the feedback provided by a very simple reinforcement function. Nevertheless, the large number of random actions taken by the robot, especially at early stages, makes this paradigm too slow and forces the use of a robot-simulator to learn the optimal control policy. In this article we suggest the use of a new approach based on the combination of a genetic algorithm, reinforcement learning, and an adjustable number of states, which makes the learning fast (Section III.A), and increases its stability (Section III.B). The mutual influence that the genetic algorithm and the reinforcement exert on each other varies along the learning process, so that the influence of the GA is high at the beginning, while the RL plays an increasing importance role with the passing of time. While the GA helps the robot to explore sequences of actions with a rapidly decreasing degree of randomness, the RL prevents the GA from being trapped in local minimums. Finally, the properties of the regular Markov chains constitute a powerful tool for adjusting the number of states so that the attention is focused only on those states that are relevant in the robot's long term behaviour. Finally, it is important to mention here that our approach shows a low dependency on the value of the learning parameters (table I), so that a reasonable selection of their values is enough to get good results.

Our proposal was used to solve two different and common tasks in mobile robotics: wall following and door traversal. The experimental results confirm a high performance of our proposal, not only as the required learning times are significantly reduced and the stability of the process is improved, but also as our proposal was able to solve a complex task, such as door traversal, when the only thing we changed was the reinforcement function.

Acknowledgment

The authors thank the support from grants TIN2005-03844, PGIDIT04TIC206011PR, TIC2003-09400-C04-03.

References

[1] C. W. Anderson A. G. Barto, R. S. Sutton. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13, 1983.

[2] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[3] P. Cichosz and J. J. Mulawka. Fast and efficient reinforcement learning with truncated temporal differences. In *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

[4] Pawel Cichosz. *Reinforcement Learning by Truncating Temporal Differences*. PhD thesis, Dpt. of Electronics and Information Technology, Warsaw University of Technology, 1997.

[5] U. Nehmzow. *Scientific Methods in Mobile Robotics*. Springer, 2005.

[6] J. Correa R. Iglesias, C. V. Regueiro and S. Barro. Improving wall following behaviour in a mobile robot using reinforcement learning. In *ICSC International Symposium on Engineering of Intelligent Systems, EIS'98*, 1998.

[7] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3, 1988.

[8] T. Kyriacou S. Billings U. Nemhzow, R. Iglesias. Robot learning through task identification. *International Journal on robotics and autonomous systems*, 54:766–778, 2006.

[9] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, England, 1989.