ρ-Learning: A Robotics Oriented Reinforcement Learning Algorithm

Josep M. Porta
Institut de Robòtica i Informàtica Industrial (UPC-CSIC)
Gran Capità 2-4, 08034, Barcelona, SPAIN
tel. +34 93 4015791 fax. +34 93 4015750
e-mail: jporta@iri.upc.es

Submission Type: SHORT PAPER

Keywords: Reinforcement Learning, Robot Learning, Sensor Relevance, Walking Robots.

Abstract

We present a new reinforcement learning system more suitable to be used in robotics than existing ones. Existing reinforcement learning algorithms are not specifically tailored for robotics and so they do not take advantage of the robotic perception characteristics as well as of the expected complexity of task that robots are likely to face. In a robot, the information about the environment comes from a set of qualitative different sensors and in the main part of tasks small subsets of these sensors provide enough information to correctly predict the effect of actions. Departing from this analysis, we outline a new reinforcement learning system that aims at determining relevant subsets of sensors for each action and we present an algorithm that partially implements this new reinforcement learning architecture. Results of the application of the algorithm to the problem of learning to walk with a six legged robot are presented and compared with a well known reinforcement learning algorithm (Q-Learning) showing the advantages of our approach.

ρ-Learning: A Robotics Oriented Reinforcement Learning Algorithm

Josep M. Porta
Institut de Robòtica i Informàtica Industrial (UPC-CSIC)
Gran Capità 2-4, 08034, Barcelona, SPAIN
e-mail: jporta@iri.upc.es

1 Introduction

Reinforcement Learning (RL) [3] is a promising approach to achieve the control of complex robots in dynamic environments. However, the adaptation of existent RL algorithms so that they can be applied to robotics is not free of problems. In this paper, instead of departing from any existent reinforcement learning algorithm, we analyze the RL paradigm departing from the special features of robotic tasks and we propose a new RL system based on the characteristics of robot perception and on the expected complexity of tasks robots are likely to face.

Within the RL framework, the learner must survey the reward derived from the execution of actions. These surveys are used to execute the actions from which more reward is expected. In the case of a robot, the reward predictions must be based on its sensor readings[1].

If the robot pays attention to all its sensors, then it can in theory predict the effects of any action as correctly as possible. But the use of all the sensor readings to predict the reward is only necessary for the most complex problems. In general, a reduced subset of sensors provides sufficient information to correctly anticipate the results of a given action. Paying attention to all sensors implies to pay attention to many non relevant signals that can be considered as noise that slows the learning process to the point of making it impractical.

Existent RL algorithms assume that the teacher provides a way to extract the relevant information to predict the effect of actions. However, in real robotic applications, the a priori knowledge of which are the relevant sensors for achieving a task is not usual. For this reason, in many robotic problems, existent RL algorithms are applied using all sensors accepting the complications derived from that but avoiding modifications in the algorithms and also avoiding the problem of determining the relevant subsets of sensors for each action. We propose the alternative approach consisting in confronting the problem of finding the relevant subsets of sensors to avoid the inconveniences of working with all the sensor readings. We believe that this is a more adequate approach in robotics since the relevant subsets of sensors are likely to have small cardinality compared with the total amount of sensor of the robot and so, working with all sensors implies to deal with a lot of noise and finding the (we assume small) relevant subsets may be not so difficult and once identified, the learning will be faster.

Any available knowledge about which are the relevant subsets of sensors for each action should be used but the teacher is only likely to be able to identify those subsets in simple cases and so, we have to assign this task to an automatic process. Obviously, a brute force approximation consisting is testing all the possible subsets of sensors is not a good idea (there are too many of them!). Since we assume that the relevant subsets of sensors have small cardinality, we propose

[1] In general, not only the current sensor readings but also the previous ones and the memory of already executed actions can be relevant to predict the reward. This information as well as the user knowledge about the task (specially relevant sensor values, ...) can be represented as *virtual sensors*. In the paper, when we refer to sensors we mean both the physical and the virtual ones (if any).

an incremental search strategy: we start examining subsets of only one sensor and if in a given situation those subsets are not useful enough to accurately predict the reward of a certain action, then subsets of higher cardinality are considered. In a degenerate case, our system will iterate until all the sensors are taken into account to predict the reward of a specially complex action. Observe that the last case considered by our learning system is the first considered one in other approaches. To implement the proposed search strategy, we need a module able to select from a collection of (initially simple) subsets of sensors the ones that are useful for any action. A complementary module will be in charge of modifying the collection of subsets of sensors when necessary. In this paper we present an algorithm that implements the first one of these two modules.

The paper is organized as follows. First, our algorithm is presented. In section 3, the algorithm is applied (in simulation) to a robotic task: we confront the task of learning to walk with a six legged robot. Next we compare our work with other similar ones and finally we extract some conclusions of our work and outline ways in which it can be extended.

2 ρ-Learning

The objective of the ρ-learning (for ***Ro***bot-Learning) algorithm is that of determining which is the subset of sensors (from an initially provided repertory) that gives a more reliable prediction of the reward of each action. So, the algorithm should learn a reward prediction for each subset of sensors and action and a confidence measure to select the most reliable of these predictions for each action.

For the reward prediction, our algorithm adjusts the expected average reward for each action and combination of values of the sensors included in each subset of sensors. In this version of the algorithm and for simplicity reasons, we store separate statistics for each action and combination of values of the subsets of sensors. However, since the reward obtained for similar sensor values is likely the be somehow related, the use of generalization techniques (i.e. clustering or function approximation techniques) would be greatly helpful to reduce the amount of stored data and to increase the convergence speed.

For the confidence measure, the algorithm estimates the average absolute error on the reward prediction described before (the lower the error the greater the confidence). If generalization techniques are used for the reward prediction, they must be also used for the error.

Formally (see algorithm 1), if G is the initially provided collection of subsets of sensors, g is a member of G, C_g is the Cartesian product of the ranges of the sensors included in g, $S(g)$ is an element of C_g and a is an action, then we define the mappings:

$$Reward_{g,a} : C_g \rightarrow \mathbb{R}$$

$$Error_{g,a} : C_g \rightarrow \mathbb{R}$$

The reward mapping should estimate

$$q(g, S(g), a) = R_{S(g),a} + \sum_{t=1}^{\infty} \gamma^t R_t^*$$

Where $R_{S(g),a}$ is the reward obtained if action a is executed when the sensors in g have values $S(g)$, R_t^* is the reward obtained t steps after the execution of a assuming that after that action the robot acts optimally and γ is a discount rate to balance the importance of future reward with respect to immediate one. This infinite addition can be expressed recursively using a Bellman-like equation:

$$q(g, S(g), a) = R_{S(g),a} + \gamma\, V(S')$$

Teacher defined inputs:

- A collection of subsets of sensors (G)
- Set of Actions (A)
- Reinforcement Signal

Inicialization:

For each subset of sensors g, readings of this group of sensors S(g) and action a

$q(g,S(g),a)=0$ and $error(g,S(g),a)=0$

Algorithm:

Do forever:

1.- For each g in G

$S(g) \leftarrow$ Subset g of the current sensor readings

2.- a $\leftarrow$ Action to be executed

a is choosen according to Q(S,j) for the sensor readings S and j in A where

$Q(S,j)=q(g,S(g),j)$

for g in G with $error(g,S(g),j) <= error(g',S(g'),j)$ (g' also in G)

3.- Execute the action and wait until its conclusion.

4.- r $\leftarrow$ Reinforcement generated by the execution of a.

5.- For each g in G

$S'(g) \leftarrow$ Subset g of the current sensor readings

6.- Update q(g,S(g),a) and error(g,S(g),a)

For each g in G

$q(g,S(g),a) \leftarrow \alpha\, q(g,S(g),a)+(1-\alpha)(r+\gamma V(S'))$

$error(g,S(g),a) \leftarrow \alpha\, error(g,S(g),a)+(1-\alpha)\left|q(g,S(g),a)-(r+\gamma V(S'))\right|$

where

$V(S')=\underset{j}{Max}[Q(S',j)]$

α is a learning rate.

γ is a discount parameter.

Algorithm 1: *The ρ-Learning algorithm.*

Teacher defined inputs:

- Set of States (S)
- Set of Actions (A)
- Reinforcement Signal

Inicialization:

Q(s,a)=0 for each state s and action a

Algorithm:

Do forever:

1.- s ← Current State

2.- a ← Action to be executed

a is choosen according the information stored in Q(s,j) for each j in A

3.- Execute the action and wait until its conclusion.

4.- r ← Reinforcement generated by the execution of a

5.- s' ← New State

6.- Update Q(s,a)

$Q(s,a) \leftarrow \alpha\, Q(s,a)+(1-\alpha)(r+\gamma V(s'))$

where

$V(s')=\underset{j}{Max}[Q(s',j)]$

α is a learning rate.

γ is a discount parameter.

Algorithm 2: *The Q-Learning algorithm.*

where $V(S')$ is a goodness evaluation of the next attained situation after executing a. This evaluation is the maximum expected reward for the actions executable in that situation.

In step 6 of algorithm 1, you can see that the reward mapping is updated so that it tends to the value indicated by the Bellman equation. Even not shown in the figure, the first obtained reward is taken directly as the value for the reward mapping and from then the shown Widrow-Hoff updating rule is used (this is a simplified MAM updating rule [9] cited in [11]). Observe that the update is performed for all the monitored subsets of sensors. In this way the executed action is evaluated from many points of view simultaneously.

The error mapping is updated every time the reward mapping is updated using the same updating rule.

In step 2, a different reward prediction for each monitored subset of sensors and each action j is obtained and the error is used to identify the most reliable ones that are used to select the next action to be executed.

We can compare our algorithm with Q-learning (algorithm 2 [10]), a well known RL algorithm. This algorithm is based on identifying the state of the environment. This state is suppose to include the information necessary to obtain a single reward estimation for each action (step 2) and is also used to update the predicted reward after the action execution (step 6). Q-learning constructs a reward mapping but it does not need any statistic about its error.

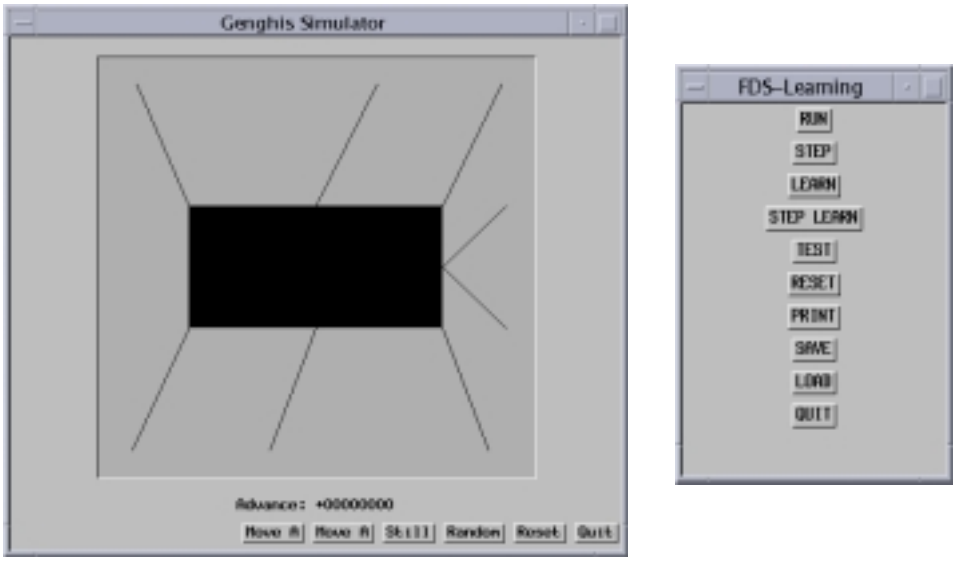

Figure 1: *The simulated six-legged robot (left) and the learning command window (right).*

The ρ-learning memory requirements are of the order

$$O(ng\, nc\, na)$$

with ng the number of the subsets to be monitored, nc the maximum cardinality of the Cartesian products C_g described before and na the number of actions. The Q-learning memory requirements are

$$O(ns\ na)$$

with ns the number of states. In many cases, ns is the total amount of possible combinations of values of all the sensors. In this case, since we consider a reduced collection of small subsets of sensors, ns is much larger than $ng\, nc$ and the result is that, in general, ρ-learning uses less memory than Q-learning.

In table 1, you can see a comparison of the complexity of the two main steps of the Q and ρ-learning algorithms. In general ρ-learning increases the times by a factor ng. Since the size of the collection of monitored subset of sensors is reduced, this cost increment is not an impediment for the online application of the algorithm.

Algorithm Step	**Q-Learning**	**ρ-Learning**
Reward prediction (2)	$O(na)$	$O(ng\, na)$
Information Update (6)	$O(1)$	$O(ng)$

Table 1: *Execution time comparison between Q-learning and ρ-learning. Numbers in the first column refer to steps in algorithms 1 and 2.*

3 Learning to Walk

3.1 Experiment Setup

To test ρ-learning, we confront (in simulation, figure 1) the task of learning to walk with a six legged robot as that of figure 2. This is not a trivial problem but we know enough about it [1] to interpret the results.

We binarize the leg position sensors in the following way:

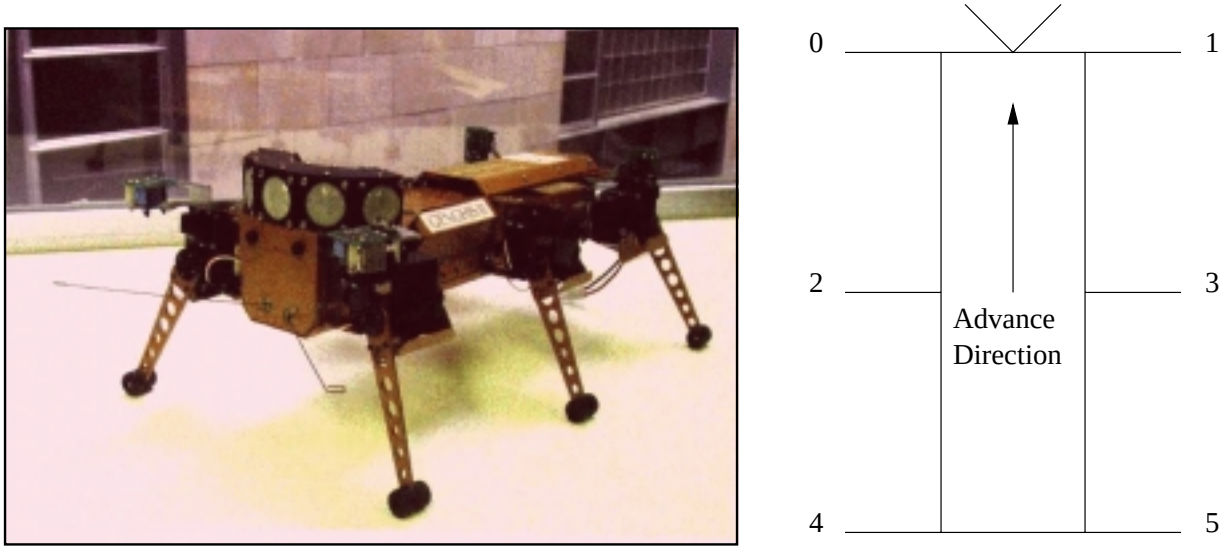

Figure 2: *The Genghis II six-legged robot and its schematic top view.*

- Vertical position sensors: With value 0 when the corresponding leg is at the ground level and 1 if not.
- Horizontal position sensors: With value 0 when the leg is more advanced than the central position of its workspace and 1 if not.

Additionally, we define a set of actions meaningful for the task to be achieved. When walking, each leg of the robot can be in two different phases:

1. **Return Stroke:** The leg is lifted and advanced.
2. **Power Stroke:** The leg is moved to the ground to support the robot body.

In our experiment, the set of actions the robot can execute can be defined using six bits, one for each leg (so we have 64 (2^6) actions). Each bit indicates if the corresponding leg should be in return stroke (value 1) or in power stroke (value 0) after the execution of the action. For instance, the action $A : 1\,0\,1\,0\,0\,0$ puts legs 0 and 2 in return stroke and the rest of legs in power stroke. An automatic user-defined mechanism (called *global alpha-balance* [1]) is in charge of the coordinated backward movement of legs in power stroke so that the robot advances.

The learning task we aim at achieving can be defined as:

> *Generate the adequate sequence of actions (or gait) so that the robot advances as much as possible within a limited time departing from an initial posture with all legs in contact with the ground but in a random advance position.*

This implies that the robot has to learn not to fall down and to coordinate the steps of the different legs to advance as fast as possible. In particular, the robot must produce the gait depicted in figure 3 called the *tripod gait* since it is known to be the fastest stable gait executable with a six-legged robot [1]. Other gaits can produce higher instantaneous advance speed but its average speed is lower than that of the tripod gait.

We set up a reinforcement signal that includes the two subtasks the robot should learn:

- **Not to fall down:** If the robot falls down, the reward it receives is -500.
- **Advance:** If an action does not produce a fall, then the reward given to the robot is the displacement of the robot produced by the action.

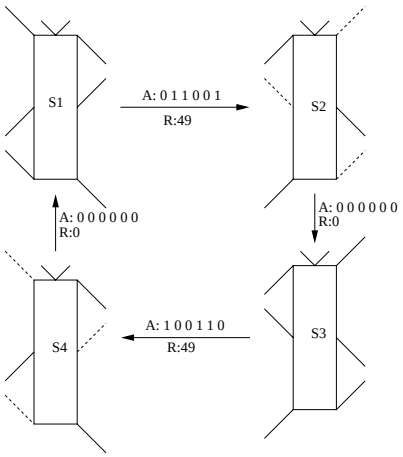

Figure 3: *In the tripod gait four different situations are encountered and only three different actions are used. In the figure, dashed legs stand for legs in the air after performing a step. Observe that action* $A : 000000$ *always gets reward 0 and so its evaluation must be based on the future rewards it allows to obtain.*

Actions are selected using a Boltzmann exploration rule in which the probability of executing and action a in a situation S is

$$p(S,a) = \frac{e^{k\ Q(S,a)}}{\sum_{\forall a'} e^{k\ Q(S,a')}}$$

where $Q(S,a)$ is determined depending on the algorithm and k is slightly increased departing from 0 until a threshold (0.75) is reached so that the best action is chosen with increasing probability. The rest of parameters were: γ 0.9, α 0.1 and we limited each experiment to at most 24 time slices. In this period and using the tripod gait, the robot can accumulate a reward between 539 and 588 depending on the random initial conditions of the experiment. If the robot falls down then the corresponding negative reward is applied and a new experiment is started. The results shown in next sections are the average of ten independent runs. In each run, the evaluations of the learning process are based on averaging the performance of the robot from 20 random initial postures.

3.2 Results

To apply ρ-learning, the teacher must define a collection of subsets of sensors. We use subsets of only one (virtual) sensor in accordance with the incremental approximation to discover the relevant subset of sensors explained in the introduction.

In figure 4, you can see the result of applying the algorithm with different increment rates of the parameter k. If k is increased too fast (so things discovered at initial phases of the learning are taken as definitive), then suboptimal solutions are attained. This is the well known [3] trade-off between exploration and exploitation typical of RL.

The algorithm identifies the best action for each situation but due to the probabilistic action selection mechanism we use, the average accumulated reward for each experiment is lower than the optimal one (observe that the best results shown in figure 4 are around 525 and as explained before, this quantity could be greater). In figure 5, you can see a typical probability assignment achieved using ρ-learning for the situations and actions included in the tripod gait.

This problem can be also confronted using Q-learning. In this case the teacher has to define a state. This can be achieved concatenating the 12 binary virtual sensors described before. This produces a space of states with cardinality 4096 (2^{12}). In figure 6 you can see the best result

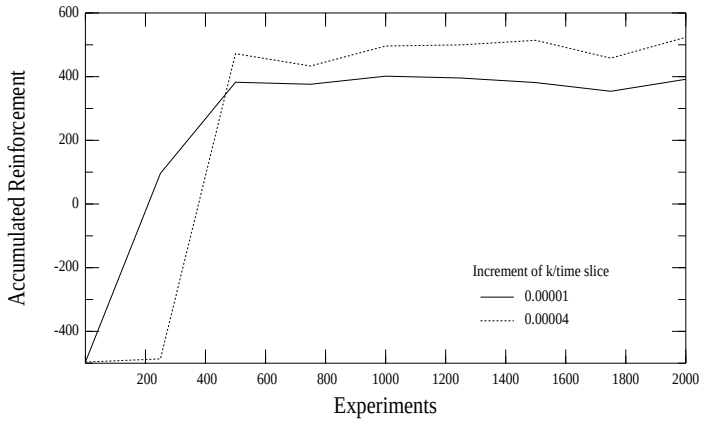

Figure 4: *The performance using ρ-learning with various parameter settings of the action selection mechanism.*

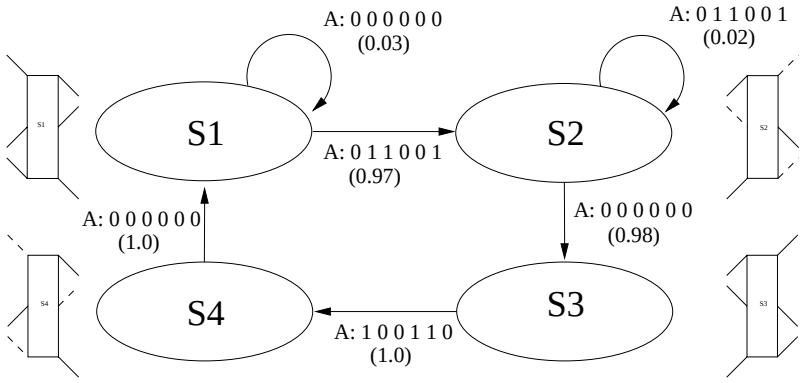

Figure 5: *Typical action probability distribution assigned by ρ-learning.*

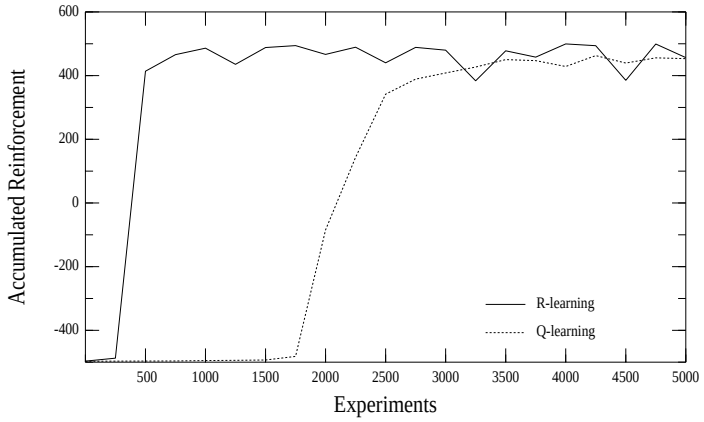

Figure 6: *ρ-learning vs Q-learning.*

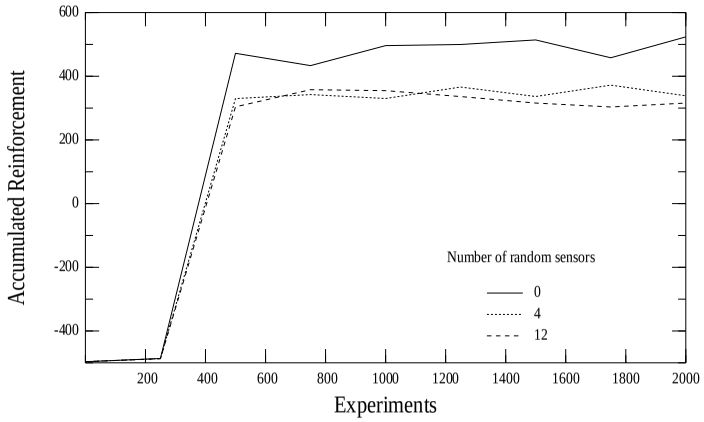

Figure 7: *The performance using ρ-learning adding random sensors.*

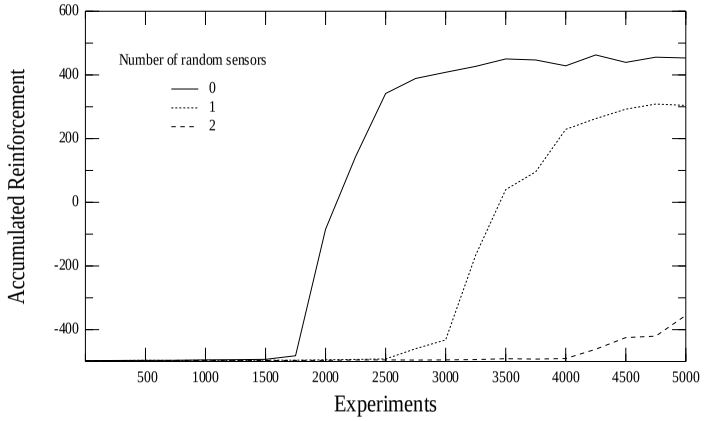

Figure 8: *The performance using Q-learning adding random sensors.*

we obtained using Q-learning compared with results using ρ-learning for the same number of experiments. Our algorithm converges to a solution equivalent to that attained using Q-learning in about five times less experiments.

As far as the memory requirements is concerned, our algorithm uses $2 \cdot 12 \cdot 64 \cdot d = 1536 \cdot d$ bytes of memory to store 2 estimations for each subset of sensors (12) and action (64) where d is the size required to store a *real* value. By its side Q-learning uses $1 \cdot 2^{12} \cdot 64 \cdot d = 262144 \cdot d$ bytes to store one *real* value for each state (2^{12}) and action (64). This supposes that our algorithm uses around 170 times less memory than Q-learning.

The performance obtained using ρ-learning can oscillate due to non relevant subsets of sensors: because of the use of a Widrow-Hoff updating rule, at a given moment a non relevant subset of sensors can acquire a temporary high relevance and the algorithm can rely on its apparently correct reward predictions to execute a non optimal action. After this mistake the relevance of the non relevant subset of sensors is decreased and the robot acts optimally again. To eliminate these oscillations we only have to purge the non-relevant subsets of sensors but this is not part of the presented algorithm.

To test the performance of our algorithm when provided with many irrelevant inputs[2] we set up an experiment in which many random binary sensors (active 50 per cent of times) are given

[2]As for instance sensors that the user believes to be interesting but that are actually useless for the task at hand.

to the algorithm. In figure 7, we can see the results when we use none, 4 and 12 random sensors with the same set of learning parameters: the performance decreases but not dramatically. Experimentally we have observed that if the exploration parameter k is adjusted specifically for each case (increasing more slowly if more noisy sensors are present) a better performance than that shown in figure 7 can be achieved.

A similar experiment was performed using Q-learning. In figure 8, you can see the performance of Q-learning with only 1 and 2 random sensors. The addition of such amount of irrelevant sensors as those we use for ρ-learning is almost impossible: if we define the state as the concatenation of input sensors, the size of the data stored increases exponentially with the number of sensors and so does the convergence time.

4 Related Work

The main part of existing RL algorithms do not work with subsets of sensors but with all the sensor readings simultaneously and try to accelerate the learning process using generalization techniques (see [6] for instance).

Our RL system can be seen as just another generalization technique since the reward predicted from a subset of sensors can be used in all situations where these sensors have the same readings independently of the readings of all other sensors. The difference is that we work with many alternative generalizations (one for each considered subset of sensors and the one that produces minimal error is used) while in other generalization methods only one way of generalizing in the space of sensors is considered at a time.

There exist other RL approaches that do not use all the sensor readings to predict the effect of actions [11]. The main difference between those approaches and ours is that they use a genetic algorithm to find out the relevant subsets of sensors while we advocate for the use of an incremental strategy that is likely to be more efficient for robotic tasks.

The problem of learning to walk with a six legged robot has been addressed by other authors before with different formalizations. In [5] specific methods based on immediate reward are used and in [7] a simplified version of the problem is used to test an algorithm able to deal with Non-Markovian spaces of states. By its side, [2] uses a learning architecture based on self-organizing neural networks and [4] proposes an evolutionary strategy to develop a neural network to control the gait of the robot.

5 Conclusions and Future Work

In this paper we have outlined a RL system that takes advantage of the special characteristics of the robotic problems so that they can be learned more efficiently. The exploitation of these special features is necessary if we aim at applying RL to control real robots.

We decompose the general problem of applying RL to a robotics in three subproblems that consists in:

1. Determining interesting subsets of sensors. To find them we propose an incremental strategy based on testing first simple subsets and test more complex ones only when necessary.
2. Finding mappings from subsets of sensors to reward predictions.
3. Determining which mapping to use in a given situation.

Our assumption is that in the case of robots, the combined difficulty of these tree problems is lower than that of finding a global mapping from all sensors to reward predictions. The experiments run until now seem to validate that assumption but there is still much work to do to confirm this impression. There are two main lines of future work:

- Improve the presented algorithm: In this line, we plan to:
  - Use generalization techniques to generate the reward and error mappings for each subset of sensors and action.
  - Define a new action selection mechanism more adequate for our formalization. This mechanism could use not only the reward prediction but also the error prediction and the number of times an action has been executed to decide whether or not that action is worth to be executed again.
  - Apply it to more complex problems.
- Extend the presented algorithm: As stated before the presented algorithm is only the kernel of a general learning system. The next module to be developed will be in charge of modifying the collection of monitored sets of sensors. The information collected for small subsets of sensors can be used to monitor only the most potentially useful subsets of higher cardinality guiding the search in the large space of subsets of sensors. With this module the problems caused by non-relevant subsets of sensors outlined in section 3.2 would be minimized and we could confront problems where more complex combinations of sensors need to be considered. Another future module of our learning system will deal with the automatic generation of virtual sensors (including previous sensor readings for instance) so that they can be considered in the learning process.

From a more general point of view, the work presented in this paper is the first step toward the relaxation of some assumptions present in usual RL formalizations [8] concerning how the environment is perceived by the robot (this is the assumption relaxed in this paper), which action model is more adequate for robotic learning and when the effects of actions are detected by the robot. New algorithms based on these relaxed assumptions are necessary to overcome the challenge of facing increasingly complex tasks with increasingly sophisticated robots.

Acknowledgements

I wish to thank the *Institut de Robòtica i Informàtica Industrial* for providing the material means for this research and Dr. Enric Celaya for his help during the realization of this work and in the revision of the paper.

References

[1] Celaya E. and Porta J.M. (1998): "A Control Structure for the Locomotion of a Legged Robot on Difficult Terrain", IEEE Robotics and Automation Magazine, Vol. 5, No. 2, Special Issue on Walking Robots.

[2] Ilg W., Mühlfriedel T. and Berns K. (1997): "A Hybrid Learning Architecture Based on Neural Networks for Adaptive Control of a Walking Machine", Proceedings of the 1997 IEEE International Conference on Robotics and Automation, pp. 2626-2631.

[3] Kaelbling L. P., Littman M. L. and Moore A. W. (1995): "An Introduction to Reinforcement Learning", In "The biology and technology of intelligent agents", Springer-Verlag, pp. 90-127.

[4] Kodjabachia J. and Meyer J. A. (1998): "Evolution and Development of Modular Control Architectures for 1-D Locomotion in Six-Legged Animats", Connection Science, 10, 211-237.

[5] Maes P. and Brooks R. A. (1990): "Learning to Coordinate Behaviors", Proceedings of the AAAI-90, pp. 796-802.

[6] Mahadevan S. and Connell J. (1992): "Automatic Programming of Behavior-based Robot Using Reinforcement Learning", Artificial Intelligence 55. pp. 311-363.

[7] Pendrith M. D. and Ryan M. R. K. (1996): "C-Trace: A New Algorithm for Reinforcement Learning of Robotic Control", In Proceedings of the International Workshop on Learning for Autonomous Robots (Robolearn-96).

[8] Porta J. M. and Celaya E. (1999): "Reinforcement Learning and Automatic Categorization", In proceedings of the 7th IEEE International Conference on Emerging Technologies and Factory Automation.

[9] Venturini G. (1994): "Apprentissage Adaptatif et Apprentissage Supervisé par Algorithme Génétique", Thèse de Docteur en Science (Informatique), Université de Paris-Sud.

[10] Watkin, C. J. C. H. and Dayan P. (1992): "Q-learning", Machine Learning 8, pp. 279-292.

[11] Wilson S. W. (1995): "Classifier Fitness Based on Accuracy", Evolutionary Computation, 3.