*International Journal of Innovative Computing, Information and Control*
Volume **10**, Number **4**, August **2014**

ICIC International ©2014 ISSN 1349-4198
pp. 1471–1494

# HIERARCHICAL NEURO-FUZZY MODELS BASED ON REINFORCEMENT LEARNING FOR AUTONOMOUS AGENTS

Karla Figueiredo[1,2], Marley Vellasco[1], Marco Pacheco[1] and Flavio Joaquim de Souza[3]

[1]Department of Electrical Engineering
Pontifical Catholic University of Rio de Janeiro
Rua Marquês de São Vicente, 225, Rio de Janeiro 22451-900, Brazil
{ karla; marley; marco }@ele.puc-rio.br

[2]Department of Applied Mathematics and Computational Science
Universidade Estadual da Zona Oeste
Rio de Janeiro 23070-200, Brazil
Karla.figueiredo@gmail.com

[3]Department of Systems and Computer Engineering
Universidade do Estado do Rio de Janeiro
Rio de Janeiro, Brazil
flavioj@gmail.com

Received May 2013; revised November 2013

Abstract. *This work introduces a new class of neuro-fuzzy systems for intelligent agents, called Reinforcement Learning – Hierarchical Neuro-Fuzzy System. This new class combines a hierarchical partitioning method of the input space with a Reinforcement Learning algorithm to achieve the following important characteristics: automatic creation of the model's structure; self-adjustment of the parameters; autonomous learning of the actions; capacity to deal with a greater number of inputs; and automatic generation of linguistic fuzzy rules. The proposed model was devised to overcome limitations of traditional reinforcement learning methods based on lookup tables, particularly in applications involving continuous environments and/or environments considered to be high dimensional. The paper details the hierarchical neuro-fuzzy architecture, its basic cell, and the learning algorithm. The performance of the proposed system was evaluated in four benchmark applications – the Mountain Car Problem, the Cart-Centering Problem, the Inverted Pendulum and the Khepera Robot Control. The results obtained demonstrate the capacity of the novel hierarchical neuro-fuzzy system to automatically extract knowledge from the agent's direct interaction with large and/or continuous environments. This knowledge is in the form of fuzzy linguistic rules, with no prior definition of the number and position of the fuzzy sets.*

**Keywords:** Reinforcement learning, Autonomous agents, Hybrid neuro-fuzzy, Hierarchical partitioning, Robotics

1. **Introduction.** This work presents a detailed description of a new hybrid neuro-fuzzy model, called Reinforcement Learning – Hierarchical Neuro-Fuzzy Politree (RL-HNFP), which is based on a Reinforcement Learning algorithm to provide an agent with intelligence, making it able, by interacting with its environment, to acquire and retain knowledge for reasoning (infer an action).

The proposed model was devised based on the analysis of the limitations of existing Reinforcement Learning (RL) models [1,2] and on the desirable characteristics for RL-based learning systems, particularly in applications involving continuous environments and/or environments considered to be high dimensional.

For continuous and/or large state space, the application of traditional Reinforcement Learning methods based on lookup tables (a table that stores value functions for a small or discrete state space) is no longer possible, since the state space becomes too large. This problem is known as the curse of dimensionality [2]. In order to bypass it, some form of generalization must be incorporated into how the states are represented [3-8].

The generalization issue in RL can be seen as a structural credit assignment problem. A structural credit assignment implies spatial propagation of costs across similar states. However, standard RL algorithms perform temporal credit assignment, which implies temporal propagation of costs [9]. Generalization is, therefore, a complex problem, since there is no 'Spatial Difference' learning algorithm, in contrast with standard Temporal Difference algorithms [2].

This generalization is usually obtained with the insertion of function approximation techniques in the RL methods. In this case, value functions are updated by reinforcements not only of the state related to the current iteration, but also of other states that are correlated due to some common characteristics [10,11].

Current models that make use of function approximation usually require some predefinitions, such as the specification of the number of layers and neurons in each layer for neural network-based models and the size of the lookup table in standard RL models. Other popular solution, the CMAC [12-16], associates different states to the same action. In order to provide greater flexibility, some studies have applied fuzzy logic to this problem, obtaining promising results [1,6,8,17-24]. However, even in this case, predefinitions are required, such as the number of fuzzy rules and the number and format of the fuzzy sets used in the fuzzy rules' antecedents and consequents.

Hasselt [6], in a recent study, confirms the trend of using fuzzy logic to provide generalization in continuous environments. He highlights, however, that "A drawback of fuzzy sets is that these sets still need to be defined beforehand, which may be difficult".

Therefore, the main objective of this work was to propose a new hybrid neuro-fuzzy model that would perform function approximation (for generalization purpose) with no predefined parameters, which enhances the agent's autonomy.

This paper is organized in four additional sections. Section 2 contains a brief description of the hierarchical Politree partition used in the proposed model. Section 3 introduces the proposed Reinforcement Learning – Hierarchical Neuro-Fuzzy Politree (RL-HNFP) model, describing its basic cells, the hierarchical architecture and complete learning process. Section 4 presents the results obtained with four case studies, chosen to compare the performance with other similar models and to evaluate the proposed model in different environments. Lastly, Section 5 presents the conclusions of this work.

## 2. **Hierarchical Partitioning.**

The partitioning process of the input/output space has great influence on the neuro-fuzzy system performance in relation to its desirable features (accuracy, generalization, automatic generation of rules, etc.).

The most common partitioning methods used by the neuro-fuzzy systems currently found in literature are: Fuzzy Grid, Adaptive Fuzzy Grid, Fuzzy Boxes and Fuzzy Clusters [25-29]. One of the constraints of these partition methods is related to the limited number of inputs allowed, due to the rule explosion problem [30]. For instance, a neuro-fuzzy system with five input variables, each with its universe of discourse divided into 4 fuzzy sets, may obtain a total of 1024 rules ($4^5$). If the number of inputs is increased to 20, and the same division of the universe of discourse is used, the result is an unmanageable total of 1,099,511,627,776 rules ($4^{20}$). The use of Fuzzy Clusters as input partition reduces this problem; however, the obtained fuzzy rules are less interpretable in this case.

Another way to minimize the curse of dimensionality problem is to make use of recursive partitionings, such as Binary Space Partitioning (BSP) [31,32], which employ recursive processes in their generation. Additionally, by preserving the independence of the input characteristics, these types of partitioning maintain the interpretability of the model in a fuzzy rule format.

The use of recursive partitioning methods in hybrid neuro-fuzzy models was introduced by Souza et al. [29] and resulted in a new class of neuro-fuzzy systems, called Hierarchical Neuro-Fuzzy Systems. The recursive partitioning was inspired by the demonstration that hierarchically structured rule bases present a linear growth in the number of rules [33].

The Politree partitioning, which is a generalization of the BSP, subdivides the $n$-dimensional space in $m = 2^n$ subdivisions, where each of the $n$ inputs can assume two linguistic values: *low* and *high*. The Politree partitioning can be represented by a tree structure, as can be visualized in Figure 1 that presents an example of a Politree partitioning with two inputs ($n = 2$). Figure 1(a) presents the resultant two-dimensional space subdivision and Figure 1(b) presents the tree format. As can be seen from Figure 1, the input space is initially divided in $m = 2^2 = 4$ partitionings. In a recursive way, partition 3 has been sub-divided in $m = 2^2 = 4$ new sub-partitionings, as the result of the applied learning algorithm (see Section 3.3).

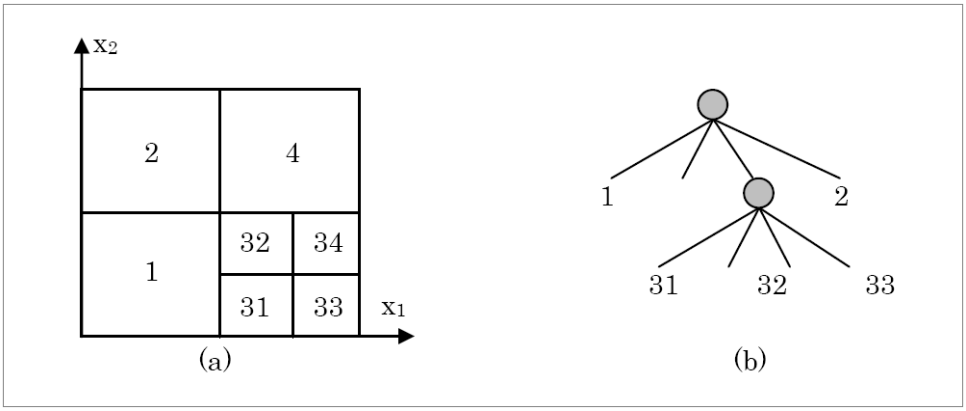


FIGURE 1. (a) Example of a Politree partitioning with 2 inputs, where $m = 2^2$ and (b) tree representation of the Politree partitioning with 2 inputs

This idea of hierarchically dividing the input space has been used in the proposed model, as detailed in the following sections.

3. **Reinforcement Learning – Hierarchical Neuro-Fuzzy Politree (RL-HNFP) Model.** The RL-HNFP model is composed of one or various standard cells called RL-neuro-fuzzy Politree (RL-NFP) [34-37]. These cells are laid out in a hierarchical structure in the form of a tree. The following sub-sections describe in detail the basic cells, the resultant hierarchical structure and the complete learning algorithm proposed.

3.1. **Reinforcement learning – neuro-fuzzy Politree cells.** An RL-NFP cell is a mini-neuro-fuzzy system that performs $2^n$ partitioning ($n$ = number of inputs variables) of a given input space in accordance with complementary membership functions, described in Figure 2(a), in each input dimension. The mathematical representation of the sigmoid membership function is described by two parameters: '$\alpha$' that defines its slope at the transition point, and '$\beta$' which defines the transition point. Each RL-NFP cell receives **all** the inputs that are being considered in the problem. Each input variable, read by one of the agent's sensors, is evaluated in the antecedents' fuzzy sets ($low - \rho(x) -$ and *high*

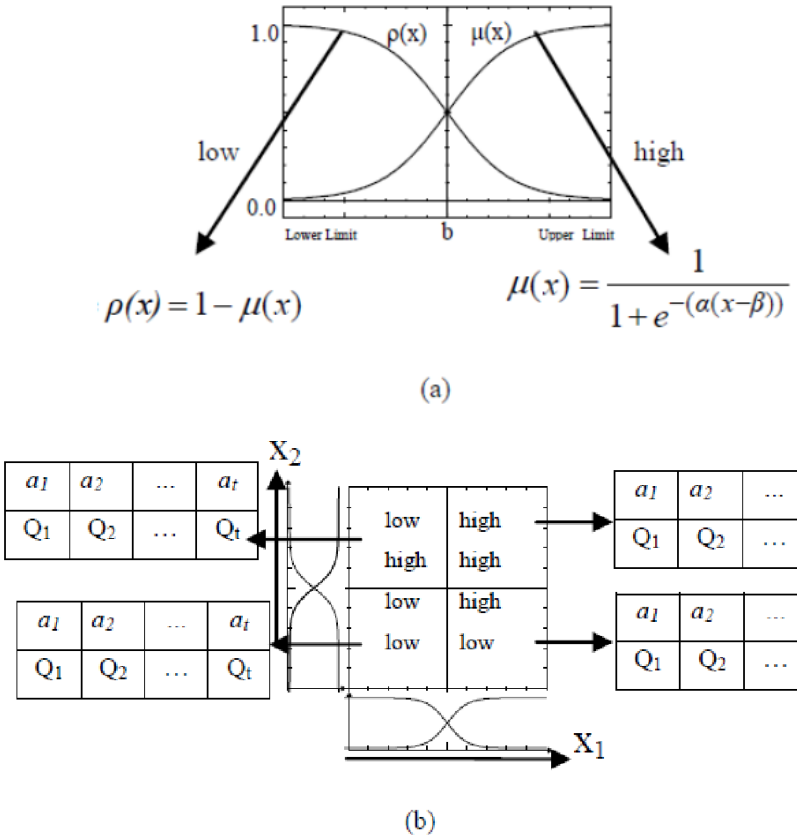


FIGURE 2. (a) Membership function representation; (b) internal representation of the RL-NFP cells with 2 input variables, where $\{a_1, a_2, \ldots, a_t\}$ is the set of available actions

$-\mu(x)$), shown in Figure 2(a). Figure 2(b) depicts an example of the resultant partition of an RL-NFP cell with two inputs ($x_1$ and $x_2$), resulting in four sub-partitions (Quadtree partitioning).

3.1.1. *Fuzzy rule representation at each RL-NFP cell.* The linguistic interpretation of the mapping implemented by the RL-NFP cell (with two inputs) depicted in Figure 2(b) is given by the following set of rules:

$\text{rule}_1$: If $x_1 \in \rho_1$ and $x_2 \in \rho_2$ then $y = a_i$
$\text{rule}_2$: If $x_1 \in \rho_1$ and $x_2 \in \mu_2$ then $y = a_j$
$\text{rule}_3$: If $x_1 \in \mu_1$ and $x_2 \in \rho_2$ then $y = a_p$
$\text{rule}_4$: If $x_1 \in \mu_1$ and $x_2 \in \mu_2$ then $y = a_q$

where each $a_i$, $a_j$, $a_p$ and $a_q$ is the chosen action from the set of possible actions available at each partition.

3.2. **RL-HNFP architecture.** A complete Reinforcement Learning – Hierarchical Neuro-Fuzzy Politree (RL-HNFP) model is created by interconnecting the basic cells described above, as illustrated in Figure 3. The cells form a hierarchical structure that results in the rules that compose the agent's reasoning. The outputs of the cells in the lower levels are the consequences of the cells in the higher levels. Figure 3(a) and Figure 3(b) exemplify an RL-HNFP architecture with $n$ inputs and, consequently, $m = 2^n$ partitions at each level.

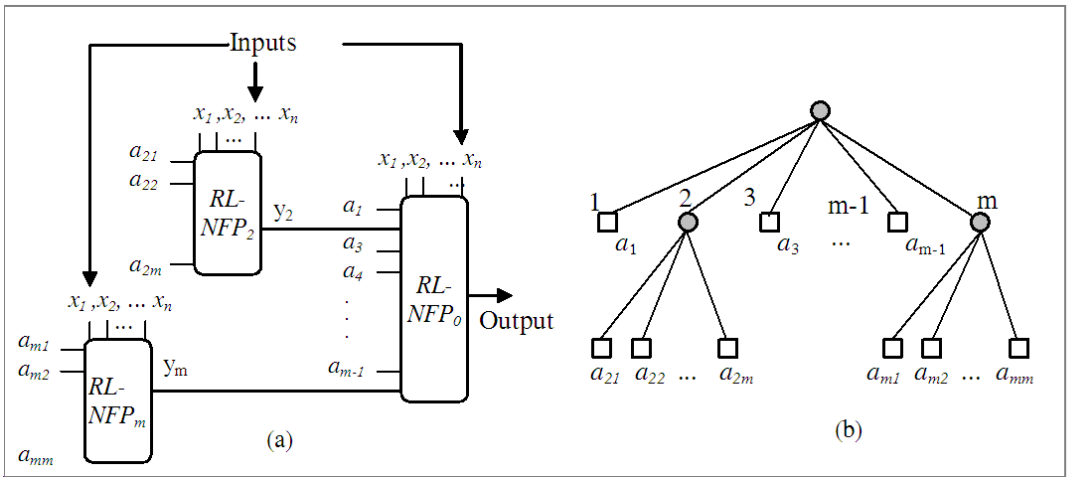


FIGURE 3. Example of an architecture of the RL-HNFP model: (a) an example of an architecture with $n$ inputs; (b) the tree format of the architecture in Figure 3(a)

The consequence of each cell's partitions corresponds to one of the two possible consequences below:

- **a singleton** (fuzzy singleton consequence, or zero-order Sugeno): *the case where* $y = constant = a_k$;

- **the output of a stage of a previous level**: *the case where* $y = y_k$, *where* $y_k$ *represents the output of a generic cell 'k', whose value is calculated by Equation (1) below. Since the high* ($\mu$ *and low* $\rho$ *membership functions are complementary (see Figure 2(a)), the defuzzification process is simplified, with the denominator of the equation on the left being equal to 1 for any values of inputs 'x';* $\omega_i$ *and* $a_i$ *correspond to the firing level and the chosen action of the i-th partition, respectively.*

$$y = \left( \sum_{i=1}^{2^n} \omega_i \times a_i \Big/ \sum_{i=1}^{2^n} w_i \right) \qquad y = \sum_{i=1}^{2^n} \omega_i \cdot a_i \tag{1}$$

Although the singleton consequence is simple, it is not previously known. Each singleton consequence is associated with an action that has not been defined a priori. Each partition has a set of possible actions $(a_1, a_2, \ldots, a_t)$, as shown in Figure 2(b), and each action is associated with a Q-value function, which is updated according to the Reinforcement Learning (RL) algorithm described in Subsection 3.3. The Q-value is defined as the sum of the expected values of the rewards obtained by the execution of action $a$ in state $s$, in accordance with a policy $\pi$. For further details about RL theory, see [2] or [38].

By means of the RL-based learning algorithm (see Subsection 3.3), one action of each partition (for example, $a_i$, $a_j$, $a_p$ and $a_q$ in the rules above) is defined as the one that represents the desired behaviour of the system whenever that system is in a given state. Thus, the consequences are the actions that the agent must learn along the process. The number of actions is not related to the number of input variables or the number of cells in the final structure.

In the architecture presented in Figure 3(a) and Figure 3(b), the poli-partitions 1, 3, 4, ..., $m-1$ have not been subdivided, having as rules' consequences the values $a_1$, $a_3$, $a_4$, ..., $a_{m-1}$, respectively. On the other hand, poli-partitions 2 and $m$ have been subdivided, so the consequences of their rules are the outputs ($y_2$ and $y_m$) of subsystems 2 and $m$, respectively. On its turn, these subsystems have, as consequence, the values $a_{21}$, $a_{22}$, ..., $a_{2m}$, and $a_{m1}$, $a_{m2}$, ..., $a_{mm}$, respectively. Each '$a_i$' corresponds to a consequence

of zero-order Sugeno (singleton), representing the action that will be identified (among the possible actions), through Reinforcement Learning, as being the most favorable for a certain state of the environment. The output of the system depicted in Figure 3(a) (defuzzification) is given by Equation (2). Again, in this equation, $\omega_i$ corresponds to the firing level of partition $i$ and $a_i$ is the singleton consequence of the rule associated with partition $i$.

$$y = \omega_1.a_1 + \omega_2 \sum_{i=1}^{2^n} \omega_{2i}.a_{2i} + \omega_3.a_3 + \omega_4.a_4 + \ldots + \omega_m \sum_{i=1}^{2^n} \omega_{mi}.a_{mi} \tag{2}$$

Figure 4(a) and Figure 4(b) exemplify a particular RL-HNFP architecture with two inputs ($n = 2$). Figure 4(b) represents the partitioning of the architecture shown in Figure 4(a). Figure 4(b) also exhibits the partitions of variables $x_1$ and $x_2$ of a particular architecture with 2 inputs, where $\rho_0(x_1)$, $\rho_0(x_2)$, $\mu_0(x_1)$ and $\mu_0(x_2)$ are related to membership functions of cell RL-NFP$_0$ and $\rho_1(x_1)$, $\rho_1(x_2)$, $\mu_1(x_1)$ and $\mu_1(x_2)$ are related to membership functions of cell RL-NFP$_1$. Note that, in this example, after the creation of cell RL-NFP$_0$, the learning process has identified that the partition 2 (relating to membership function *low* for input variable $x_1$ and *high* for input variable $x_2$) must be sub-partitioned to attain better specification of the state space. Therefore, a new cell (RL-NFP$_1$) is automatically created that recursively sub-divides the second partition into four new partitions (partitions 21, 22, 23 and 24), corresponding to a sub-domain of the initial $x_1$ and $x_2$ universe of discourses.

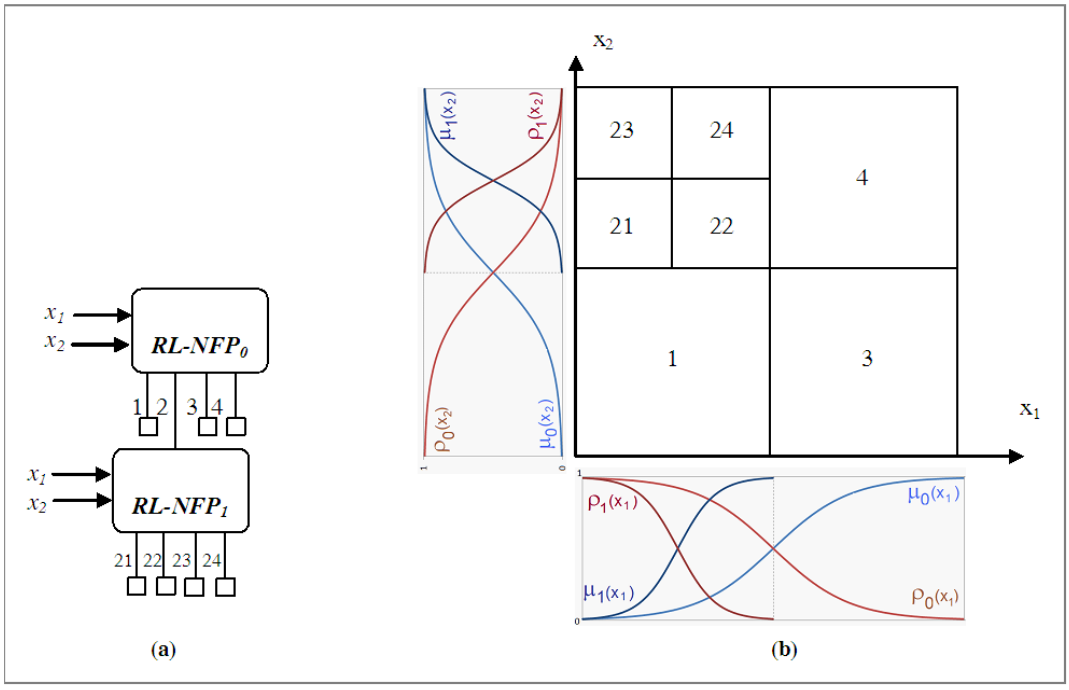


FIGURE 4. Example of an architecture of the RL-HNFP model: (a) cells format of the architecture with 2 inputs; (b) partitioning of the variables $x_1$ and $x_2$, $\rho_0(x_1)$, $\rho_0(x_2)$, $\mu_0(x_1)$ and $\mu_0(x_2)$ are related to membership functions of cell 0 and $\rho_1(x_1)$, $\rho_1(x_2)$, $\mu_1(x_1)$ and $\mu_1(x_2)$ are related to membership functions of cell 1 of Figure 4(a)

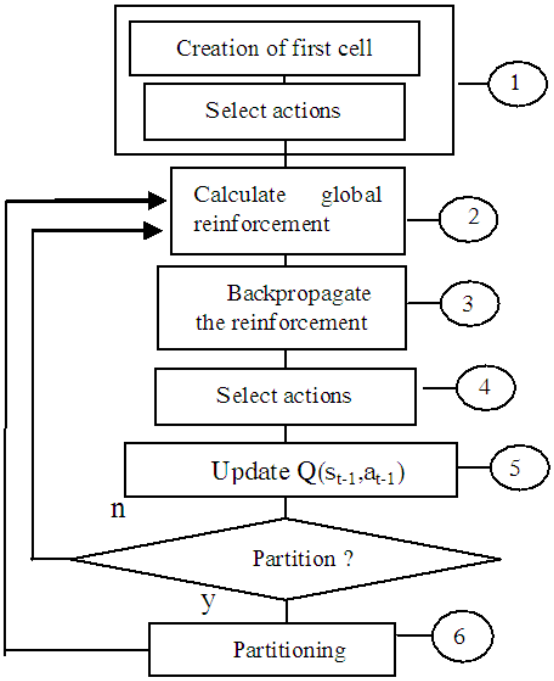


FIGURE 5. Learning algorithm of the RL-HNFP model

The hierarchical characteristic of the proposed model smoothens the value function generalization process (with the hierarchical fuzzy tree as the function approximator) without affecting the results, which is a good requirement for process convergence [2].

3.3. **RL-HNFP learning algorithm.** Neuro-fuzzy learning processes are generally divided into two parts: structure identification and parameter adjustment. The RL-HNFP model performs these two tasks in a single algorithm. The flowchart shown in Figure 5 describes the complete learning algorithm of the RL-HNFP model, which consists of six main stages that are described below.

The learning process starts out with the definition of the input variables that are relevant to the system/environment in which an agent is inserted and of the set of actions it may use in order to attain its objectives. The initial Q-values associated with the actions must also be defined a priori. Applications that make use of SARSA or Q-Learning normally start their Q-values with zero [2]. The proposed RL-HNFP learning process is based on the SARSA method, as described in Stage 5. Therefore, all Q-values are initialized with zero value.

The agent must run many cycles to ensure correct learning in the system/environment where it is inserted, as in any other model based on RL. A cycle is defined by the number of steps the agent takes in the environment from the point where the agent has been initialized to the point considered as being its goal. Each stage comprises the complete execution of the algorithm, from the agent's reading of the input variables to performing the action.

**Stage 1 – Generate root-cell (or father-cell):**

A root-cell is created with fuzzy sets whose domain is the range of possible values of the selected input variables. The value of the cell's input variables is read from the environment, normalized and applied to the cell's inputs. Then, the (normalized) $x$ values are evaluated in the *low* and *high* fuzzy sets, which result in two membership degrees, $\rho(x)$ and $\mu(x)$, respectively, for each input variable $x$. Each poli-partition chooses one of

the actions (from its set of actions), based on the methods described in Stage 4 of this algorithm. The cell output, representing the action that will be executed by the agent's actuator, is calculated by the defuzzification process given by Equation (1).

**Stage 2 – Calculate global reward:**

After the action is carried out, the environment is read once again. This reading allows the calculation of the global reward value from the environment, which will be used to evaluate the action taken by the agent. This value must be calculated by means of an evaluation function defined according to the agent's objectives. Hence, this evaluation function defines not only the nature of the reinforcement, by rewarding or punishing $(+1/-1)$, but also its intensity, thereby favoring a more efficient process in guiding this agent during the learning process. This evaluation function is application-dependent and four examples are provided in the Case Studies presented in Section 4.

Although there are cases where the agent receives a reward only at the end of each cycle (delayed reward) [2,38], the proposed RL-HNFP model is based on a learning process that implements the reward at each learning step. In this case the agent's actions determine not only its next state in the environment, but also the immediate reward. As mentioned in [39], having a reward every time is not a restriction, since the classic delayed reward problem can be considered as a special case with most rewards equal to zero.

**Stage 3 – Backpropagate the reward:**

At each step, the local reward is calculated for each partition of all active cells by means of their participation in the resulting action. In this way, the global reward calculated by the evaluation function is backpropagated from the root-cell to the leaf-cells, and weighted by the firing level of each respective leaf-cell. The firing level ($\omega_i$) is calculated using the T-norm operator. Figure 6 below presents an example of RL-HNFP architecture with two input variables $x_1$ and $x_2$, where only partitions 1 and 4 of the root cell have been hierarchically partitioned. The firing levels of the root cell RL-NFP$_0$ and its descendants RL-NFP$_1$ and RL-NFP$_4$ are calculated by Equations (3), (4) and (5), respectively, and their local rewards are defined by Equations (6), (7) and (8) ($\mathrm{R}_{\mathrm{global}}$ is the calculated global reward).

$$\begin{aligned}
\omega_{01} &= \rho_0(x_1).\rho_0(x_2) \\
\omega_{02} &= \rho_0(x_1).\mu_0(x_2) \\
\omega_{03} &= \mu_0(x_1).\rho_0(x_2) \\
\omega_{04} &= \mu_0(x_1).\mu_0(x_2)
\end{aligned} \tag{3}$$

$$\begin{aligned}
\omega_{11} &= \rho_1(x_1).\rho_1(x_2) \\
\omega_{12} &= \rho_1(x_1).\mu_1(x_2) \\
\omega_{13} &= \mu_1(x_1).\rho_1(x_2) \\
\omega_{14} &= \mu_1(x_1).\mu_1(x_2)
\end{aligned} \tag{4}$$

$$\begin{aligned}
\omega_{41} &= \rho_4(x_1).\rho_4(x_2) \\
\omega_{42} &= \rho_4(x_1).\mu_4(x_2) \\
\omega_{43} &= \mu_4(x_1).\rho_4(x_2) \\
\omega_{44} &= \mu_4(x_1).\mu_4(x_2)
\end{aligned} \tag{5}$$

$$\begin{aligned}
\mathrm{R}_{01} &= \omega_{01}.\mathrm{R}_{\mathrm{global}} \\
\mathrm{R}_{02} &= \omega_{02}.\mathrm{R}_{\mathrm{global}} \\
\mathrm{R}_{03} &= \omega_{03}.\mathrm{R}_{\mathrm{global}} \\
\mathrm{R}_{04} &= \omega_{04}.\mathrm{R}_{\mathrm{global}}
\end{aligned} \tag{6}$$

$$\begin{aligned} R_{11} &= \omega_{11}.R_{01} \\ R_{12} &= \omega_{12}.R_{01} \\ R_{13} &= \omega_{13}.R_{01} \\ R_{14} &= \omega_{14}.R_{01} \end{aligned} \tag{7}$$

$$\begin{aligned} R_{41} &= \omega_{41}.R_{04} \\ R_{42} &= \omega_{42}.R_{04} \\ R_{43} &= \omega_{43}.R_{04} \\ R_{44} &= \omega_{44}.R_{04} \end{aligned} \tag{8}$$

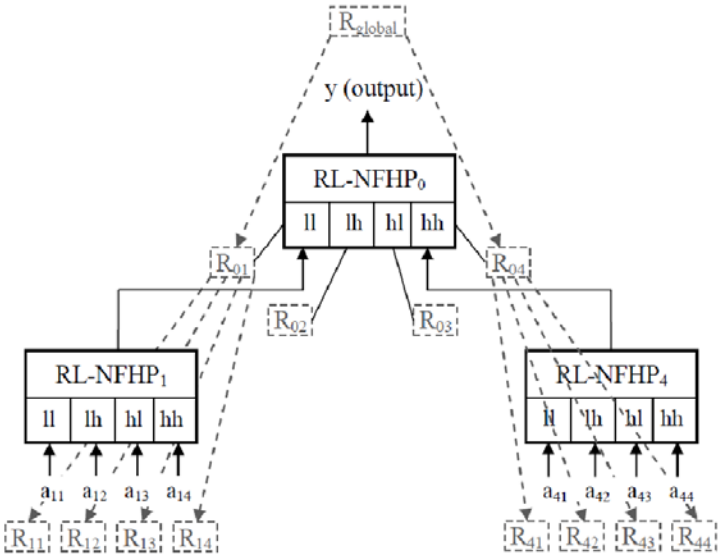


FIGURE 6. Backpropagation of the global reward of the environment for modeling RL-HNFP

**Stage 4 – Select actions:**

Each poli-partition contains a set of actions that are selected during the learning process. The selection of an action is based on the exploitation/exploration policies [2]. Usually, an agent chooses actions that are related to previously evaluated good rewards, that is, it tries to exploit what has already been learned (exploitation). However, to find better actions, it is necessary to search for alternatives that might be better than the ones already experimented, that is, the agent needs to explore the environment (exploration). Exploring state space is essential for discovering actions that correspond to the best agent's response (towards achieving an objective) when the agent is in a given state of the environment. Therefore, each action has an associated value function and makes use of the method called $\in$ – greedy [2], which selects an action associated with the highest expected Q-value with $1- \in$ probability (greedy policy) and, with $\in$ probability, it randomly selects any action (non-greedy policy). The maximum value of $\in$ is 0.1 (10%), as suggested in [2].

**Stage 5 – Update Q-values:**

Based on the values of the rewards calculated for each cell in the structure, the Q-values associated with the actions that have contributed to the resulting action carried out by the agent must be updated. The objective is to reward the value of Q by updating it according to Equation (9).

$$Q(s_t, a_t) = (1 - \alpha_t).Q(s_t, a_t) + \alpha_t[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})] \tag{9}$$

where the Q-value $(s_t, a_t)$ (of an active cell at time $t$) is updated based on its current value $Q(s_t, a_t)$; the immediate local reward $r_{t+1}$ (this is the reward calculated in Stage 3); a parameter $\gamma$ that specifies a percentage of the contribution of the Q-value associated with the next action $a_{t+1}$ chosen $(Q(s_{t+1}, a_{t+1}, ))$ when the system is in state $s_{t+1}$; and $\alpha_t$ which is the parameter that is proportional to the relative contribution of this local action within the global action (that is, the ratio between local and global reward).

Besides the update of Q-values, there is an adjustment in the rates of $\in$ – greedy (exploration/exploitation) parameter. This update is based on the analysis of the current and previous global reward:

I. **global reward value is higher than the previous global reward value** $(R_{global,t+1} > R_{global,t})$ ⇨ In this case, there is a reduction in the $\in$ – greedy exploration rates that are associated with each partition (set of actions) of each active cell.

II. **global reward value is lower than or equal to the previous global reward value** $(R_{global,t} \geq R_{global,t+1})$ ⇨ The values of the $\in$ – greedy parameters of the partitions involved are, in this case, increased, which allow different actions to have more chances of being chosen when this partition is active again, providing better exploration of the environment.

**Stage 6 – Partitioning:**

In the proposed RL-HNFP model, for a cell to be partitioned, a poli-partition must satisfy two criteria: *Growth Variable Criterion* and *Value Function Variation Criterion*. The first criterion prevents the structure from growing as a result of a bad performance caused by a still immature choice of actions (that is, it is supposed that the size of the structure might be adequate but the actions are not yet properly tuned); the second criterion encourages partitioning when there are significant variations in the actions' value functions. In other words, there are two main problems in the learning process: a) the structure might be still too small (few cells) and therefore does not answer correctly in all states (structure's cells); b) the actions associated with each cell have not yet been properly explored. Therefore, the main objective of the two proposed criteria (Growth Variable and Value Function Variation) is to control the learning process, avoiding an excessive growth of the structure. These two criteria are detailed below.

3.3.1. *Growth variable criterion.* For the first condition, a growth variable and a growth function were created so as to allow or prevent structure growth. The growth variable measures how big the variation of the Q-value has been during a certain period and is constantly compared with a pre-determined growth function. When the growth variable becomes higher than the value obtained from the growth function, the first criterion for partitioning is satisfied.

The growth variable is calculated as follows. As the cells are being updated, the Q-value variation percentage ($\Delta$Q) of the associated actions is checked. When the variation of the Q-value associated with the action is greater than a percentage of the highest variation that has occurred for this cell partition, this poli-partition is considered to present growth potential and the growth variable is increased; otherwise, it is reduced. A high Q-value variation may indicate that the actions being taken in the poli-partition in question are not suitable for the sub-domain relative to this poli-partition, so the cell must be sub-divided.

The growth function, which has been defined heuristically, may be a constant [40] or it may be a function of the number of steps, the size of the structure (tree depth), or any other function related to the learning process or objective. In the case studies presented in this paper, the growth function is a function of the number of steps and of the number of cycles, as shown in Figure 7. Thus, whenever the growth variable is greater than the

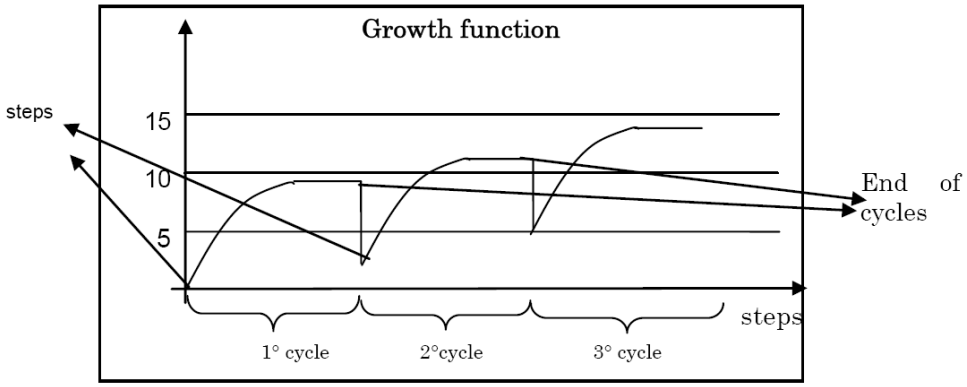


FIGURE 7. Chosen growth function to the case studies

value evaluated in the growth function (given the number of steps and cycles that the algorithm has performed at that point), the partition will satisfy the first requirement for performing the partitioning.

Ideally, the growth function must be less demanding in the beginning of the learning, (which allows the cells to sub-divide more quickly in the beginning of the learning process) and, as the system evolves, the growth function becomes more demanding, allowing the structure to grow only with higher values of the growth variable. As mentioned before, the function presented in Figure 7 was defined heuristically and used in the experiments carried out in the case studies. However, any other function with similar behavior could also have been used.

3.3.2. *Value function variation criterion.* In addition to the previous criterion, the dispersion in the $\Delta$Q-value (difference between $Q_{t+1}$ and $Q_t$) is evaluated along the learning process. A large dispersion of $\Delta$Q-value may indicate that the environment is not yet divided as necessary, making it hard to find a good Q-value for that poli-partition. Therefore, the mean and standard deviation of the $\Delta$Q-values are calculated and whenever the mean absolute value is higher than twice the standard deviation, the second criterion that determines whether or not a cell must be partitioned is satisfied. Hence, structure growth occurs according to the following two criteria:

I. the growth variable is higher than the growth function; and
II. $|\mu| > 2\sigma$, where $\mu$ and $\sigma$ are the average and the standard deviation of the $\Delta$Q-values.

When a poli-partition has all the necessary requirements for partitioning, a leaf cell is created and connected to that poli-partition. Its domain will be the sub-domain that corresponds to its closest ancestor. Leaf cells also inherit the set of actions with its respective Q-values from this ancestor.

These six stages are performed until a sufficient number of cycles for the exploration of the environment are reached. This proposed model has been evaluated in four different benchmark applications, as detailed in the following section.

4. **Case Studies.** Two versions of the hierarchical neuro-fuzzy structure have been implemented and evaluated: the RL-HNFP, where all input variables are considered in each cell, and a particular case, called RL-HNFB, where each cell allows only one input. In this case, the specification of each cell's input variable ($x_i$) is determined by a heuristic procedure (for example, considering the variables that are more important), where a certain order of variables is chosen to be presented at each level of the hierarchy. When there

are more levels than input variables, the order of variables is repeated, starting from the first variable used.

This section presents all tests that have been carried out with these models, using four different benchmarks: the Mountain Car Problem [10]; the Car-Centering Problem [41]; Inverted Pendulum [1]; and Khepera Robot Control [42]. The case studies chosen, although apparently simple, are benchmarks of the RL area, used in many RL related papers [1,2,10,41,43]. These case studies have as main objective to demonstrate that the proposed models are able to carry out autonomous learning in continuous environments with a minimum of information.

The following sections describe the benchmark applications and present the results obtained when compared with other models available in the literature. Preliminary results of the Mountain Car Problem and Khepera Robot Control have been presented in [34-37].

In all experiments presented in the following sections, the value defined for parameter $\gamma$ of the Q-value update equation (Equation (9)) was 0.9 and the initial value of parameter $\in$ was established at 0.1 the moment the cell was created. The increment/decrement rate of this parameter, when the action was respectively a penalty or a reward, was 5%. These values were defined heuristically, but follow Sutton's [11] recommendations.

4.1. **Mountain car problem.** The mountain car (see Figure 8) is a highly relevant benchmark that has been used by different researchers [1,10,11,43] with the purpose of testing their learning algorithms. The problem may be described as follows: a car must be able to get to the top of a mountain; however, the car is not powerful enough to overcome the force of gravity. Thus, in order to achieve its objective, the car must begin by moving in the opposite direction of the target so that it may add the acceleration of gravity to its own acceleration (Figure 8).

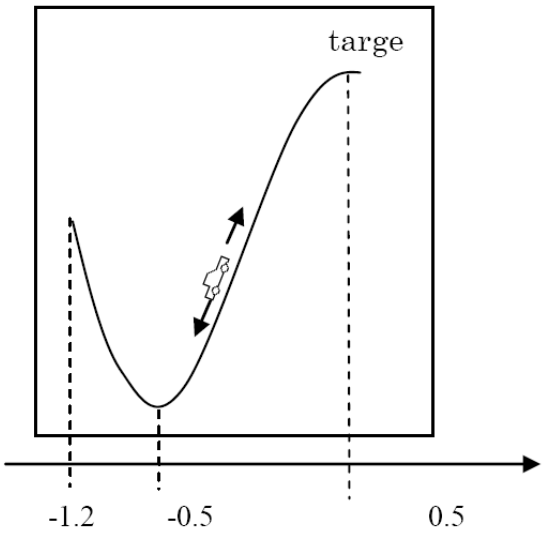


FIGURE 8. Mountain car problem

4.1.1. *Global reward calculation.* The calculation of the global reward considers the distance between the car and the objective (the mountain top), and the car's velocity. Since the car first needs to move in the opposite direction of the target, the evaluation function increases as the car gets closer to the objective (Equation (10)) or distances itself from the objective (Equation (11)). In either case, the evaluation function grows when the velocity module value also grows.

If $(x_t < x_{t+1})$, then

$$\mathrm{R}_{\mathrm{global}} = k_1 e^{(-\mathrm{distance\_objective})} + k_2 e^{(|\mathrm{velocity}|)} \tag{10}$$

If $(x_t > x_{t+1})$, then

$$\mathrm{R}_{\mathrm{global}} = k_1 e^{-(1-\mathrm{distance\_objective})} + k_2 e^{(|\mathrm{velocity}|)} \tag{11}$$

Parameters $k_1$ and $k_2$ are constants greater than 1 and are used for adapting the reinforcement values to the model.

4.1.2. *Mountain car results.* Tests were carried out with variations in the initial conditions (position and velocity) and considering different action sets. The purpose of these tests was to evaluate the performance of the models in several different situations.

Table 1 summarizes the results obtained, for different types of settings, during the learning and testing phases for both models: RL-HNFB (one input per cell) and RL-HNFP (all inputs per cell). Each row in the table is the average of the results obtained in 10 independent experiments for each of the parameters settings.

As can be seen from Table 1, each model was evaluated in four different configurations, varying the initial position and velocity, as well as the set of available actions. In the first two settings of each model, at each cycle, the car started out alternately from $x = -0.5$ (position in the mountain "valley") or $x = -1.2$ (position at the opposite end of the objective to be reached by the car) at velocity zero; in the last two settings of each model, at each cycle, the car started out randomly from any position in the interval $[-1.2, 0.5]$ at any velocity between $[-0.07, 0.07]$.

Three different sets of actions were defined: in definition A1, the set of actions is equal to $\{-1, 0, 1\}$, and the action carried out by the agent is exactly the action value calculated at the structure's output by the defuzzification process (Subsection 3.2); action set A2 is the same one that was used in A1 ($\{-1, 0, 1\}$), but the resulting action value after the defuzzification process is submitted to a limiting function. The purpose of this function is to turn into discrete the output value in the following cases: if the value at the structure's output is between 0.1 and 1, the value applied to Equation (9) is 1; if it is between $-0.1$ and $-1$, the value applied is $-1$, and if the value is between $-0.1$ and 0.1, the value calculated during the defuzzification is actually used. In set A3, the actions that are available to each cell belong to the $\{-10, -5, -1, 0, 1, 5, 10\}$ set; after the defuzzification, the upper and lower limits of the structure's output value are fixed at 1 and $-1$, respectively. The tests with conditions A2 and A3 were intended for comparison with other existing models, whose outputs are discrete.

The number of cycles needed for learning, that is, for the car to reach its objective, is higher when the initial positions are randomly selected ($5000 \times 3000$). The structure's size is also larger in this configuration, but fewer steps are necessary when the learning starts at random points in the state space. When, at each cycle, the car starts from different points in the state space, growth is stimulated in different points of the structure, generating, on average, bigger structures. On the other hand, the initial position equals $-0.5$ and velocity equals zero which is the most adverse condition for learning, resulting in more stages in the learning process with the alternate initial position approach.

The final hierarchical structure of the best result in terms of number of steps among the 10 experiments during learning was used for testing 1000 times with random initial position and velocity. The results presented in Table 1 show that both models proved to have good generalization capability, reaching the goal in few steps even in 1000 different randomly selected initial configuration.

Even with a less favourable set of actions for solving the problem (A1 action set), the learning process was effective. Set A1 is considered less favourable because it is hard for

TABLE 1. Configurations of the RL-HNFB and RL-HNFP models applied to the mountain car problem

| Learning Settings | | | | | Results | | |
|---|---|---|---|---|---|---|---|
| | | | | | | Average steps | |
| Model | Initial Position | Initial Velocity | Actions | No. of Cycles | Size of the structure | learning phase | testing phase |
| RL-HNFB | alternate $-1.2$ $-0.5$ | 0 | A1 = $\{1, 0, -1\}$ | 3000 | 198 | 230 | 98 |
| | alternate $-1.2$ $-0.5$ | 0 | A2 = $\{1, 0, -1\}$* | 3000 | 149 | 113 | 79 |
| | random $-1.2$, 0.5 | random $-0.07$, 0.07 | A2 = $\{1, 0, -1\}$* | 5000 | 237 | 90 | 86 |
| | random $-1.2$, 0.5 | random $-0.07$, 0.07 | A3 = $\{-10, -5, -1, 0, 1, 5, 10\}$** | 5000 | 179 | 131 | **71** |
| RL-HNFP | alternate $-1.2$, $-0.5$ | 0 | A1 = $\{1, 0, -1\}$ | 3000 | 121 | 221 | 101 |
| | alternate $-1.2$, $-0.5$ | 0 | A2 = $\{1, 0, -1\}$* | 3000 | 63 | 151 | 85 |
| | random $-1.2$, 0.5 | random $-0.07$, 0.07 | A2 = $\{1, 0, -1\}$* | 5000 | 136 | 100 | 91 |
| | random $-1.2$, 0.5 | random $-0.07$, 0.07 | A3 = $\{-10, -5, -1, 0, 1, 5, 10\}$** | 5000 | 96 | 91 | **69** |

* using a discretization function at the output

** the impulse applied (resulting action) is not higher than 1 or lower than $-1$.

the agent to optimize all the actions that must be learned by each partition of each cell and succeed in generating outputs from the structures near the module of 1. Therefore, the agent tends to take a greater number of steps to achieve its objective. When the chance of having the impulse value (1 or $-1$) at the output is increased using sets A2 and A3, the agent needs fewer steps in order to reach its objective.

A3 set of actions yielded the best result for both RL-HNFB and RL-HNFP, since it contains a greater number of actions as well as actions with higher absolute value, being more likely that the integral value of the impulse – never greater than $|1|$ – will be obtained at the output.

Figures 9 and 10 illustrate the best performance obtained with the RL-HNFB model and RL-HNFP, respectively, in terms of the average number of steps (shown in grey in Table 1). It may be observed from the graphs that when the car starts from the valley (position $-0.5$), it needs to oscillate from one side to the other so as to gain enough momentum to overcome the force of gravity and reach the "mountaintop". On the other hand, when the car starts from the farthest position (position $-1.2$), it already has enough potential energy to reach the objective without oscillating.

Table 2 compares the results obtained with RL-HNFB and RL-HNFP models and those provided by different RL-based models described in [18]. The number of parameters row relates to: the number of neurons in the hidden layer for the Neural Q-learning case; the number of grids for the CMAC (Cerebellar Model Articulation Controller) [23] case; and the number of rules for the Fuzzy Q-Learning (FQL) [1]. Each of these parameters is predefined in the respective models. The average number of steps in learning and testing phases are presented for each model, with the car starting out from any $x \in [-1.2, 0.5]$ and $v \in [-0.07, 0.07]$.

Table 2 shows that the worst performance is provided by the Neural Q-learning implementation. This can be explained by the fact that the Neural Q-learning approach has to learn its state perception in addition to the Q-values. For the CMAC and FQL models the state perception is fixed a priori. Furthermore, the weight modifications in

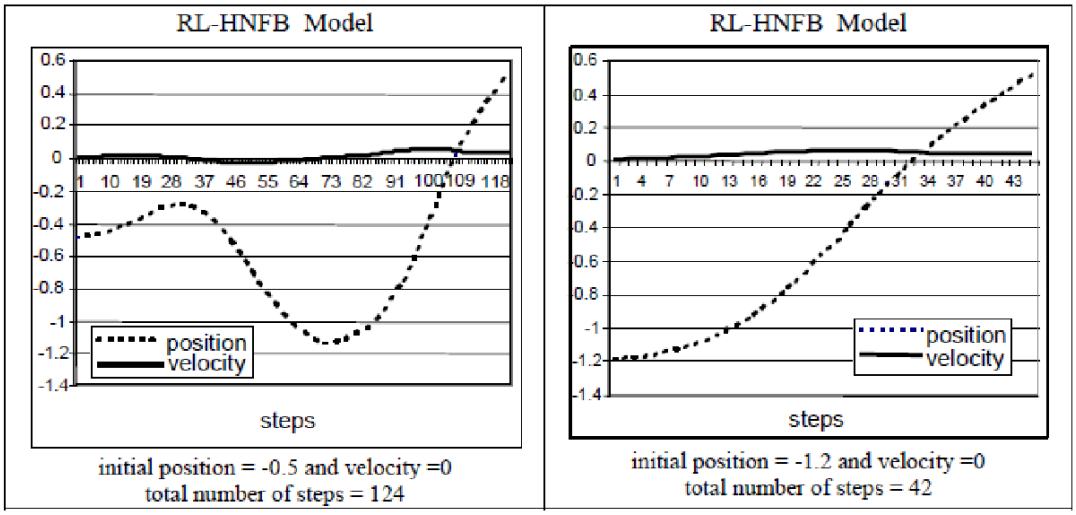


FIGURE 9. Test results for the mountain car problem with the best RL-HNFB configuration

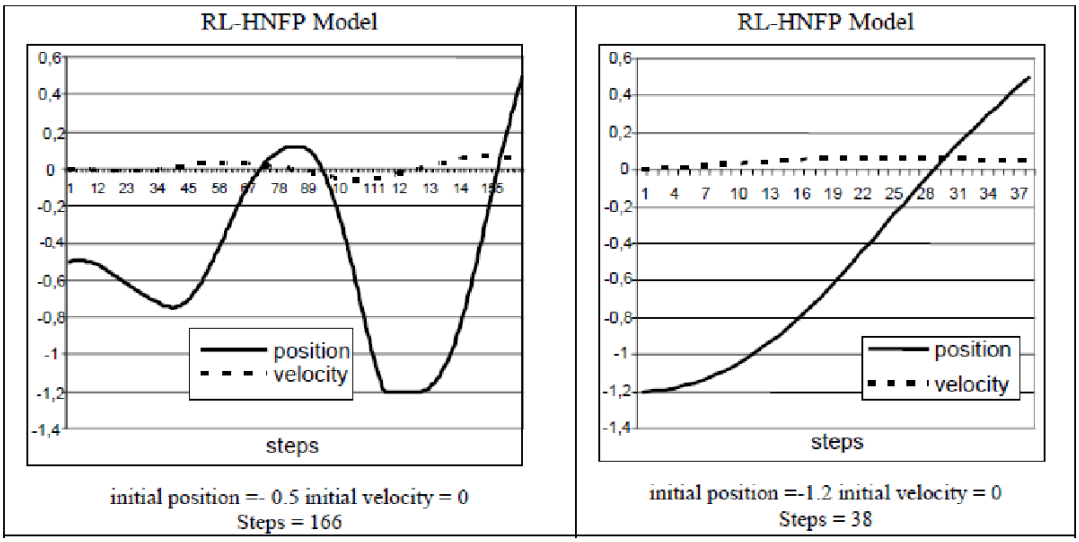


FIGURE 10. Test results for the mountain car problem with the best RL-HNFP configuration

TABLE 2. Performance comparison of the RL-HNFB and RL-HNFP models with other models [18] for the mountain car problem

| Model | No. of parameters | Learning Phase | Testing Phase |
|---|---|---|---|
| **Neural Q-learning** | 4 | 1724 | 2189 |
| **CMAC Q-learning** | 343 | 262 | 85 |
| **FQL** | 25 | 112 | 61 |
| **RL-HNFB** | 0 | 131 | 71 |
| **RL-HNFP** | 0 | 91 | 69 |

the Neural Q-learning process, accomplished by the backpropagation algorithm, affect the entire neural network, whereas the modifications in the other approaches affect only eligible rules/boxes, related to the models FQL and CMAC, respectively. Therefore, the Neural Q-learning model is not able to successfully learn the necessary policy [1].

In the case of the CMAC model, the active state set is perceived discretely (many neighbouring states activate the same grid sets); on the other hand, in the case of the FQL, RL-HNFB and RL-HNFP models the state set is perceived continuously. This explains why the results of the FQL, RL-HNFB and RL-HNFP models are better.

Although the FQL performed a little better than the RL-HNFP model in the testing set, it should be considered that like the CMAC, the FQL needs predefined information (fuzzy rules and fuzzy sets) relative to the learning process. It is important to emphasize that in the RL-HNFB and RL-HNFP models, no a priori information about the partitioning or about the rule base is provided.

It is also important to point out that the number of characteristics in the RL-HNFB and RL-HNFP models is considered to be zero in Table 2 because although the initial fuzzy sets used by the first cell were pre-defined, they are subdivided a number of times that is not pre-determined at the beginning of the learning process. The subdivisions and consequently the final configuration of fuzzy sets are performed automatically by the learning algorithm.

4.2. **Cart-centering problem.** The Cart-Centering Problem (parking the car) is generally used as a benchmark of the area of evolutionary programming, where the force that is applied to the car is of the "bang bang" type [41]. This problem was used mainly for the purpose of evaluating how well the RL-HNFB and RL-HNFP models would adapt to changes in the input variables domain without having to undergo a new training phase.

The problem consists of parking, in the centre of a one-dimensional environment, a car with mass $m$ that moves along this environment due to an applied force $F$. The input variables are the position ($x$) of the car, and its velocity ($v$). The objective is to park the car in position $x = 0$ with velocity $v = 0$. The equations of motion are:

$$x_{t+\tau} = x_t + \tau \cdot v_t \text{ and } v_{t+\tau} = v_t + \tau \cdot F_t/m \tag{12}$$

where $\tau$ represents the time unit. The global reward is calculated by Equation (13) below:

If ($x > 0$ and $v < 0$) or ($x < 0$ and $v > 0$)

$$\mathrm{R_{global}} = k_1 e^{(-|\text{distance\_objective}|)} + k_2 e^{(|\text{velocity}|)} \tag{13}$$

Else $$\mathrm{R_{global}} = 0.$$

The evaluation function increases as the car gets closer to the centre of the environment with velocity zero. The $k_1$ and $k_2$ coefficients, as in the previous case, are constants greater than 1 used for adapting the reward values to the model's structure. The values used for time unit and mass were $\tau = 0.02$ and $m = 2.0$, respectively, as used in [41].

The stopping criterion is achieved when the difference between the velocity and the position value in relation to the objective ($x = 0$ and $v = 0$) is smaller than 5% of its universe of discourse. Table 3 shows the average of the results obtained in 5 experiments for each configuration.

The proposed model was evaluated in two different environments, with different size limits imposed to the state variables during learning and testing, although, at each cycle, the car's starting points were always $x = -3$ or $x = 3$. Two sets of actions have also been evaluated in these experiments: F1 = $\{-150, -75, -50, -30, -20, -10, -5, 0, 5, 10, 20, 30, 50, 75, 150\}$ or F2 = $\{-150, -75, 0, 75, 150\}$. The number of cycles was fixed at 1000.

As can be observed from Table 3, RL-HNFP model results in a smaller structure. This was already expected, since each cell receives both input variables, while in the RL-HNFB model, a different input variable is applied at each level of the BSP tree.

As can be observed from Table 3, the broader the position and velocity limits are (in this case equal to $|10|$), the more difficult it is to learn. When a smaller set of available actions (F2) is used, the search space for the best action is reduced; however, it generates

TABLE 3. Configurations and results of the RL-HNFB and RL-HNFP models applied to the cart-centering problem

| Model | Position Limits | Velocity Limits | Set of Available Actions* | Average Size of the Structure | Average Steps Learning Phase |
|---|---|---|---|---|---|
| RL-HNFB | $\|10\|$ | $\|10\|$ | F1 | 195 cells | 424 |
| | $\|3\|$ | $\|3\|$ | F1 | 340 cells | 166 |
| | $\|3\|$ | $\|3\|$ | F2 | 293 cells | 268 |
| RL-HNFP | $\|10\|$ | $\|10\|$ | F1 | 140 cells | 221 |
| | $\|3\|$ | $\|3\|$ | F1 | 251 cells | 145 |
| | $\|3\|$ | $\|3\|$ | F2 | 193 cells | 186 |

*F1 = $\{-150, -75, -50, -30, -20, -10, -5, 0, 5, 10, 20, 30, 50, 75, 150\}$; F2 = $\{-150, -75, 0, 75, 150\}$

very abrupt variations in velocity, generating more car oscillations around the central point and, consequently, more learning steps.

Table 4 below presents the results obtained for one of the 5 experiments carried out at each configuration shown in Table 3 when the car also started out at points $\{-2, -1, 1, 2\}$, which were not used in the learning phase. Table 4 confirms the generalisation capacity of the RL-HNFB and RL-HNFP models, demonstrating that they converge even when the car starts out at different points from the ones used in the learning phase.

TABLE 4. Testing results of the proposed models applied to the cart-centering problem

| Model | Initial Position | | |
|---|---|---|---|
| | $\|3\|$ | $\|2\|$ | $\|1\|$ |
| | Average number of steps | | |
| RL-HNFB | 387 | 198 | 141 |
| | 122 | 80 | 110 |
| | 108 | 122 | 102 |
| RL-HNFP | 190 | 166 | 99 |
| | 109 | 97 | 112 |
| | 96 | 124 | 68 |

Figure 11 presents graphs of the testing performance with one of the 5 experiments carried out with the best configurations of RL-HNFB and RL-HNFP models (grey line in Table 3).

In order to evaluate the capacity of the autonomous agent to adapt to changes in the environment without requiring a new learning process, a different test was carried out with the best configurations of RL-HNFB and RL-HNFP in which the environment limits were expanded to 10 and the velocity limits were maintained at 3. The car was positioned in three different initial points: at the environment limits (10 and $-10$), halfway in the environment ($-5$ and 5) and close to the objective ($-1$ and 1). In all cases the car was able to achieve its objective, although the environment is three times larger than the environment used for learning. This was possible due to the normalization of the input variables.

The analysis of Tables 3 and 4 leads to the conclusion that the RL-HNFB and RL-HNFP models were able to park the car in the central point of the environment and demonstrates that the expected behaviour had been learned in all different settings that were tested.

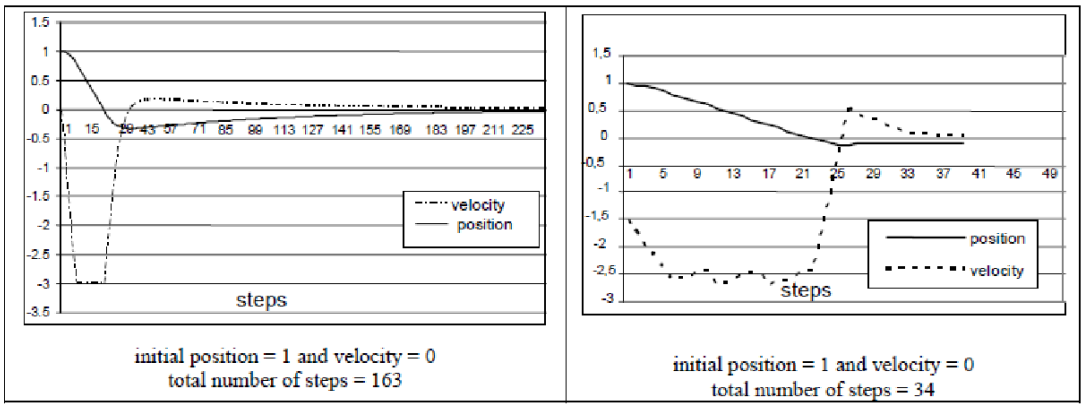


FIGURE 11. Test results with a larger environment (−10 to 10) for the cart centering problem for the (a) RL-HNFB2 configuration and (b) RL-HNFP2 configuration

4.3. **Inverted pendulum.** The Inverted Pendulum system is a classic control problem. It is a suitable process to test prototype controllers due to its high non-linearity and lack of stability. This means that standard linear techniques cannot model the nonlinear dynamics of the system. An Inverted Pendulum is a physical device consisting of a bar free to oscillate around a fixed pivot. The pole may move only in the vertical plane of the cart and track and the controller can apply an impulsive force to the cart at discrete time. The goal of the experiment is to stabilize the pendulum (bar) on the top vertical position.

This problem has double objective: try to balance the pole through the force applied to the cart, and do not move itself away from the center of the environment.

The Inverted Pendulum considers 4 input variables:

$\theta$: angle of the pole with the vertical;

$\omega$: pole angular velocity;

$x$: position on the track;

$v$: car velocity.

The dynamics of the cart-pole system are modelled by the following nonlinear differential equations [1]:

$$\ddot{\theta} = \frac{g\text{sen}(\theta) + \cos(\theta)\left[\frac{-F - ml\omega^2\text{sen}(\theta) + \mu_c\text{sgn}(v)}{m_c+m}\right] - \frac{\mu_p\omega}{ml}}{l\left[\frac{4}{3} - \frac{m\cos^2(\theta)}{m_c+m}\right]}$$
$$\ddot{x} = \frac{F + ml[\omega^2\text{sen}(\theta) - \ddot{\theta}\cos(\theta)] - \mu_c\text{sgn}(v)}{m_c + m} \tag{14}$$

where:

$g$: acceleration due to gravity = 9.8m/s$^2$;

$m_c$: mass of the cart = 1.0kg;

$m$: mass of the pendulum (pole) = 0.1kg;

$l$: the half pole length = 0.5m;

$\mu_c$: friction coefficient of cart on the track = 0.0005;

$\mu_p$: the friction coefficient of pole on the cart = 0.000002;

$\theta$: pendulum angle from vertical;

$\omega$: angular speed;

$\ddot{\theta}$: angular acceleration;

$x$: cart position coordinate;
$v$: speed of the car;
$\ddot{x}$: acceleration of the car;
$F$: force applied to the cart.

The force is applied at each 0.02s interval of time. The cycle finishes when the number of steps exceeds 50000 time steps, $|\theta| < 12$ degrees or if $|x| < 2,4$ m. The set of actions used for learning was $\{-10, -5, 0, 5, 10\}$. The evaluation function for the inverted pendulum is calculated as follows:
IF (angle >= 0 and car_velocity >= 0 and angular_velocity <= 0) or IF (angle <= 0) and (car_velocity <= 0) and (angular_velocity >= 0)
THEN

$$\mathrm{R} = k_1 e^{(-|\mathrm{angular\_}distance|)} + k_2 e^{(-|\mathrm{position}|)} \tag{15}$$

ELSE

$$\mathrm{R} = 0$$

The evaluation function increases as the angle and car position gets closer to zero.

Similar to the previous cases, $k_1$ and $k_2$ coefficients are constants greater than 1 used for adapting the reward values to the model's structure.

Because the number of inputs in the inverted pendulum is greater than the previous cases, only the RL-HNFP model was evaluated. At the end of the learning process, the structure of the RL-NFHP model presented 780 cells. This number of cells results in 1560 parameters that were tuned automatically by the learning algorithm.

The RL-NFHP model was compared with Fuzzy Actor Critic Learning model (FACL) [1]. This model has the following pre-defined configuration: 54 fuzzy rules, 3 triangular membership functions for the angular input, 3 trapezoidal membership functions for the angular speed and position, and two additional triangular membership functions for the speed. Therefore, the state space is previously mapped.

Both models have been tested with initial conditions equal to zero for the angular position ($\theta$), angular speed ($\omega$), car position ($x$) and car speed ($v$). FACL model executed its tests in a similar form as RL-NFHP model, except for the fact that the former has fixed the number of steps in 500.000. In this way, the cycle finished when it reached the goal or when the system failed due to the angle or the position exceeding the boundary values. The necessary number of cycles for the learning phase using FACL model was 60 and for the RL-NFHP 15000 cycles were necessary. Considering that the RL-NFHP model does not require any previous knowledge (number of rules or fuzzy sets), the result obtained by the RL-NFHP model was satisfactory.

Figure 12 shows the graphs for the test phase after the 15000 cycles of the learning phase. These graphs depict only 5000 steps (for better visualization). It can be noticed that the pendulum oscillates inside the interval $(-0,006; 0,002)$ and the car oscillates around the central point between the limits $(-0,002; 0,004)$ (Figure 12(a)). The angular speeds of the pendulum and of the car also are inside of the allowed limits (Figure 12(b)).

Figures 13(a) and 13(b) present the dynamic of the car and the pendulum (normalized to allow better visualization), respectively, with 350 steps extracted from Figure 12.

The main objective of this example was to evaluate the performance of the proposed model in different and more complex environments. It can be verified from the previous graphs that the model was able to control the inverted pendulum, preventing it to fall.

4.4. **Khepera robot control.** As in the Inverted Pendulum case study, only the RL-HNFP model has been evaluated since the number of input variables is bigger than the first two cases. The RL-HNFP model, together with a Khepera robot simulator [42], was

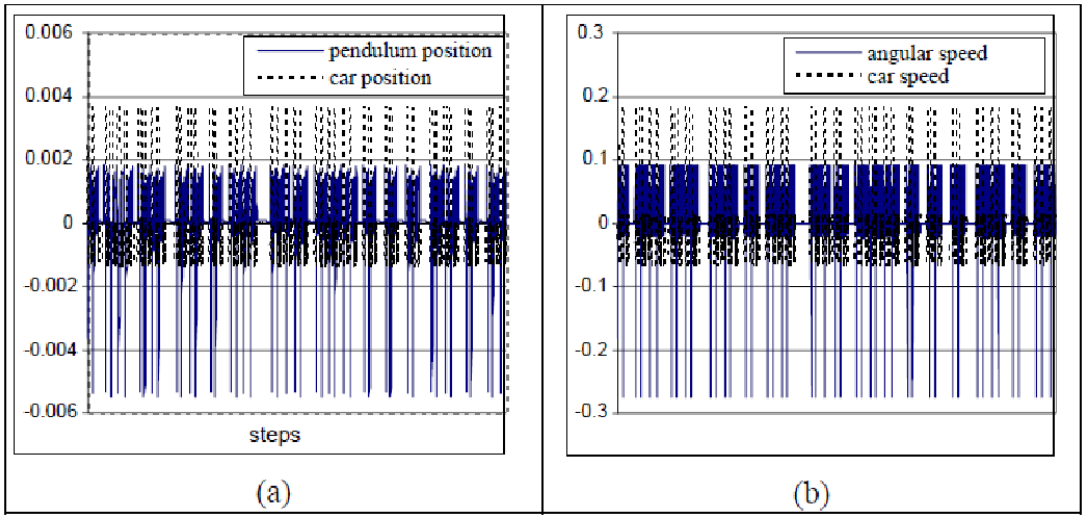


FIGURE 12. The tests results with pendulum using model RL-HNFP

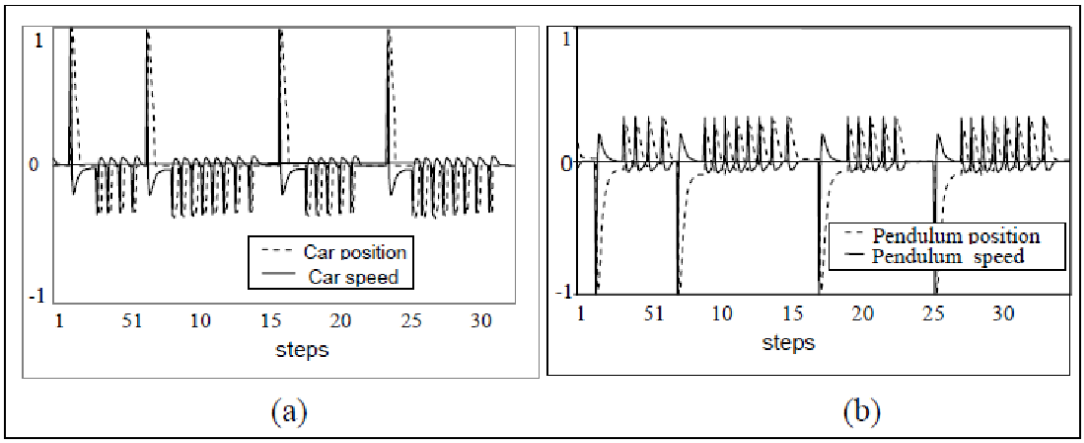


FIGURE 13. Dynamics of the car/pendulum with model RL-HNFP

tested in a squared environment where the agent should move from one of the corners and reach the diametric opposite corner.

The Khepera robot acquires information from the environment using 8 sensors grouped into four sensing variables: one ahead, one in each side and one behind. Its actions are executed via two independent motors with power varying between $-20$ and 20, one in the right side and the other in the left side. The robot scheme can be seen in Figure 14.

The simulator is composed of the robot and the navigation environment, defined by limits (walls) and by obstacles that might exist or not. It provides the robot's position $(x_r, y_r)$ and the angle $\alpha$ between the front of the robot and $x$ axis (Figure 15). These data are calculated in accordance with robot's rotation and translation movements as well as the chosen action. Angles $\beta$ and $\gamma$ (Figure 15) between the front of the robot and the goal position and between x axis and the goal, respectively, are calculated from the information given by the simulator and are used along the process.

Sensors sensibility is defined by means of exponential functions that consider the normalized Euclidian distance between sensor and obstacle. Since it is a simulator, not a robot in a real space, some other important issues, such as avoiding inappropriate obstacle surpassing or ambient limits overtaking must also be addressed.

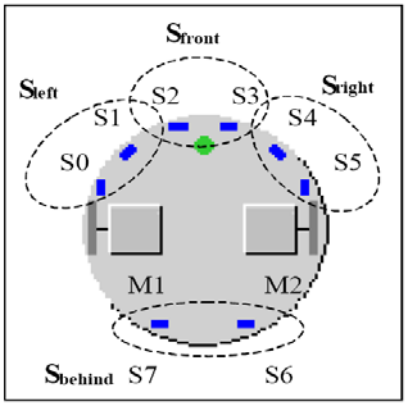


FIGURE 14. Khepera robot

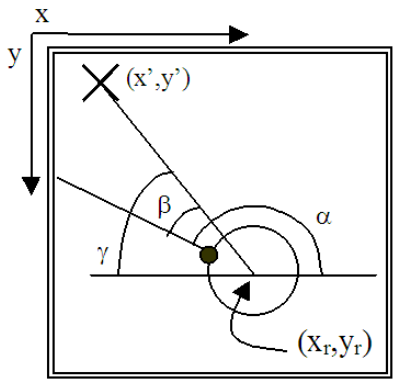


FIGURE 15. Angles definition; $(x_r, y_r)$ = robot's position

The global reward function was based on the evaluation function given by Equation (16). The evaluation function increases as the robot gets closer to the target of the environment with angles closer to zero (to motivate the robot, preferably, to use the front movement) and decreases when the robot gets closer to obstacles. In Equation (16), the value 1024 is used to normalize the value obtained by the sensor, which varies between 0 and 1023. The objective of this function (relative evaluation to the obstacle) is to reduce the reward value globally when the robot gets closer to the obstacle. Again, parameters $k_1$ and $k_2$ are constants greater than 1.

$$\mathrm{R} = \left(k_1 e^{-(\text{distance\_target})} + k_2 e^{-(|\text{distance\_angular}|)}\right) * (1 - \text{distance\_obstacles}/1024) \qquad (16)$$

The proposed model was tested in two different configurations: an environment without obstacles and with a central obstacle. The results obtained are described in the following sub-sections.

4.4.1. *Environment without obstacles.* One of the tests carried out without obstacles is shown in Figure 16, where the black dot represents the front of the robot. The learning process was performed with the robot starting at position $(-2, -2)$ and with angles 0° and 90°. Tests, on the other hand, were performed in different situations: position 1 with angle = 45°, position 2 with angle = 45°, position 3 with angle = 90°, position 4 with angle = 90°, position 5 with angle = −135° and position 6 with angle = −90°. The robot's objective was to reach position $(2, 2)$. It can be observed, for instance, that the second and fourth positions lead the robot directly to the target position, demonstrating that the acquired knowledge was generalized and that the obtained results conform to the expectative.

4.4.2. *Environment with central obstacle.* Using the same RL-HNFP structure generated using only the initial position $(-2, -2)$ with angles 0° and 90°, tests were performed with a central obstacle, also starting from different positions: position 1 with angle = 45°, position 2 with angle = 45°, position 3 with angle = −90°, position 4 with angle = −90°, position 5 with angle = 90°, position 6 with angle = 0°, and position 7 with angle = 0°. Once more, the results indicate the good performance of the proposed model in avoiding obstacles in the environment.

Table 5 presents the average number of cycles and the average number of cells of the RL-HNFP model. In this example, it is important to stress that in this case study the environment is more complex, with a much bigger state space than the others tests.

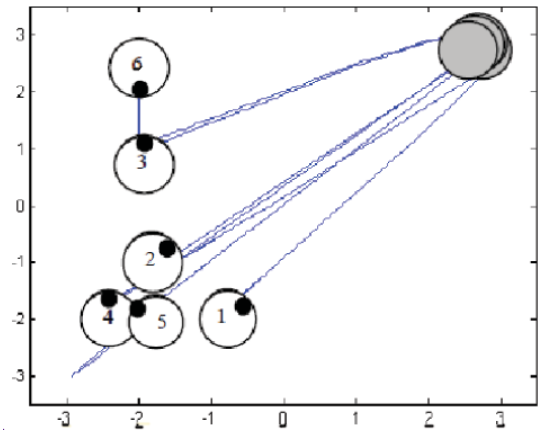


FIGURE 16. Tests without obstacles

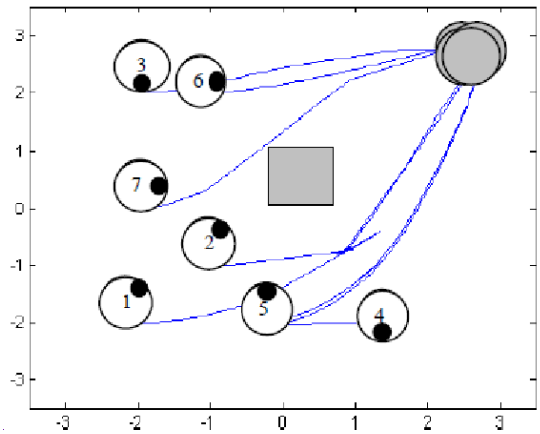


FIGURE 17. Central obstacle tests

TABLE 5. RL-HNFP model

| | **Without Obstacle** | | **Central Obstacle** | |
|---|---|---|---|---|
| | *Cycles* | *Cells* | *Cycles* | *Cells* |
| RL-HNFP | 5000 | 498 | 10000 | 812 |

Even though the RL-HNFP model has not been submitted to a real robot, it seems that the necessary computational effort would be small and the process very fast, once a set of hierarchical rules was extracted from the generated structure at the end of the learning procedure. In this way, at each step, the robot could refer to these rules and execute the resulting action.

5. **Conclusions.** The objective of this paper was to present a new neuro-fuzzy model which aims to improve the weak points of conventional neuro-fuzzy systems and reinforcement learning models. The RL-HNFP models belong to a new class of neuro-fuzzy systems called Reinforcement Learning Hierarchical Neuro-Fuzzy Systems. These models are able to create their own structures and allow the extraction of knowledge in a fuzzy rule-base format without any a priori definition of the input space partitioning.

As demonstrated by the four case studies presented, the RL-HNFP models provide the following important features: are able to create and expand the structure of rules without any prior knowledge; extract knowledge from the agent's direct interaction with large and/or continuous environments (through Reinforcement Learning); and to produce interpretable fuzzy rules, which compose the agent's intelligence to achieve his goal(s). The agent was able to generalize its actions, showing adequate behaviour when the agent was in states whose actions had not been specifically learned. This capacity increases the agent's autonomy. Normalizing the inputs also enabled the model to adequately respond to changes in the limits of input variables. This characteristic was also very interesting, because it further increases the autonomy desired for this agent.

In spite of the good results obtained by the proposed model, a few points still need improvement. Some modifications, relative to the learning algorithm, might be implemented to obtain a faster learning, a more compact structure and a lower computational cost. In future the authors intend to execute tests with Eligibility Traces [2]. This is a method that does not update only the current state function value, but also the function value of previous states inside a predefined limit.

Additionally, the RL-NFHP model can be improved using the WoLF principle (Win or Learn Fast) [44] to modify the learning rate that adjusts the policy. The WoLF principle

consists of learning quickly when the agent is losing and slowly when it is winning. Finally, the proposed model is being applied to real robots, to evaluate its performance.

## REFERENCES


[1] L. Jouffe, Fuzzy inference system learning by reinforcement methods, *IEEE Trans. on SMCs-C*, 1998.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998.

[3] M. Kamal, J. Murata and K. Hirasawa, Task-oriented reinforcement learning for continuous tasks in dynamic environment, *Proc. of the 41st SICE Annual Conference*, vol.2, pp.829-832, 2002.

[4] J. F. Peters and C. Henry, Reinforcement learning with approximation spaces, *Journal Fundamenta Informaticae*, vol.71, nos.2-3, pp.323-349, 2006.

[5] P. Wawrzynski and A. Pacut, Model-free off-policy reinforcement learning in continuous environment, *Proc. of the International Joint Conference on Neural Networks*, pp.1091-1096, 2004.

[6] H. Hasselt, Reinforcement learning in continuous state and action spaces, *Reinforcement Learning: State of the Art*, 2012.

[7] W. Sun, X. Wang and Y. Cheng, Reinforcement learning method for continuous state space based on dynamic neural network, *Proc. of the 7th World Congress on Intelligent Control and Automation*, Chongqing, China, 2008.

[8] L. Busoniu, D. Ernst, B. Schutter and R. Babuska, Continuous-state reinforcement learning with fuzzy approximation, *Adaptive Agents and Multi-Agent Systems III Adaptation and Multi-Agent Learning*, pp.27-43, 2008.

[9] C. H. C. Ribeiro, A tutorial on reinforcement learning techniques, *International Joint Conference on Neural Networks*, 1999.

[10] A. Moore, Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state-spaces, *Machine Learning: Proc. of the 8th International Conference*, Morgan Kaufmann, 1991.

[11] R. S. Sutton, Generalization in reinforcement learning: Successful examples using sparse coarse coding, *Advances in Neural Information Processing Systems*, vol.8, pp.1038-1044, 1996.

[12] R. M. Kretchmar and C. W. Anderson, Comparison of CMACs and radial basis functions for local function approximators in reinforcement learning, *Int. Conference on Neural Networks*, 1997.

[13] H. Min, J. Zeng and R. Luo, Fuzzy CMAC with automatic state partition for reinforcement learning, *Proc. of the 1st ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, pp.12-14, 2009.

[14] J. C. Santamaría, R. S. Sutton and A. Ram, Experiments with reinforcement learning in problems with continuous state and action spaces, *Adaptive Behavior*, vol.6, no.2, pp.163-218, 1998.

[15] J. S. Albus, A new approach to manipulator control, The cerebellar model articulation controller (CMAC), *Journal of Dynamic Systems, Measurement and Control*, vol.97, no.3, pp.220-227, 1975.

[16] J. S. Albus, A theory of cerebellar function, *Mathematical Biosciences*, vol.10, pp.25-61, 1971.

[17] H. R. Berenji, Fuzzy Q-learning for generalization of reinforcement learning, *Proc. of the 5th IEEE Int. Conference on Fuzzy Systems*, New Orleans, LA, USA, pp.2208-2214, 1996.

[18] H. R. Berenji and S. K. Saraf, Competition and collaboration among fuzzy reinforcement learning agents, *Proc. of the IEEE Int. Conf. on Fuzzy Systems, IEEE WCCI*, pp.622-627, 1998.

[19] A. Bonarini, A. Lazaric, F. Montrone and M. Restell, Reinforcement distribution in fuzzy Q-learning, *Fuzzy Sets and Systems*, vol.160, pp.1420-1443, 2009.

[20] M. J. Er and C. Deng, Online tuning of fuzzy inference systems using dynamic fuzzy Q-learning, *IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics*, vol.34, no.3, pp.1478-1489, 2004.

[21] A. Ferdowsizadeh, *Advanced Multi-Agent Fuzzy Reinforcement Learning*, Master Thesis, Dalarna University, Sweden, 2004.

[22] P. Y. Glorennec, Fuzzy Q-learning and dynamic fuzzy Q-learning, *Proc. of the 3rd IEEE International Conference on Fuzzy Systems*, Orlando, FL, USA, vol.1, pp.474-479, 1994.

[23] P. Y. Glorennec and L. Jouffe, Fuzzy Q-learning, *Proc. of the 6th IEEE International Conference on Fuzzy Systems*, Barcelona, Spain, pp.659-662, 1997.

[24] L. Jouffe, Ventilation control learning with FACL, *Proc. of the 6th IEEE International Conference on Fuzzy Systems*, pp.1719-1724, 1997.

[25] A. Abraham, Adaptation of fuzzy inference system using neural learning, in *Fuzzy System Engineering: Theory and Practice, Studies in Fuzziness and Soft Computing*, N. Nedjah (ed.), Germany, Springer Verlag, 2005.

[26] J. S. R. Jang, ANFIS: Adaptive-network-based fuzzy inference system, *IEEE Trans. on System Man & Cybernetics*, vol.23, no.3, pp.665-685, 1993.
[27] R. Kruse and D. Nauck, NEFCLASS – A neuro-fuzzy approach for the classification of data, *Proc. of the ACM Symposium on Applied Computing*, pp.26-28, 1995.
[28] P. Vuorimaa, Fuzzy self-organizing map, *Fuzzy Sets and Systems*, vol.66, pp.223-231, 1994.
[29] F. Souza, M. Vellasco and M. Pacheco, Hierarchical neuro-fuzzy QuadTree models, *Fuzzy Sets & Systems*, vol.130, no.2, pp.189-205, 2000.
[30] M. Vellasco, M. Pacheco, L. S. Ribeiro Neto and F. Souza, Electric load forecasting: Evaluating the novel hierarchical neuro-fuzzy BSP model, *Electrical Power and Energy Systems*, vol.26, pp.131-142, 2004.
[31] N. Chin and S. Feiner, Near real-time shadow generation using BSP trees, *Proc. of Association for Computing Machinery's Special Interest Group on Computer Graphics and Interactive Techniques*, vol.23, no.3, pp.99-106, 1989.
[32] Y. Chrysanthou and M. Slater, Computing dynamic changes to BSP trees, *Proc. of European Association for Computer Graphics*, vol.11, pp.321-332, 1992.
[33] M. Brown, K. M. Bossley, D. J. Mills and C. J. Harris, High dimensional neurofuzzy systems: Overcoming the curse of dimensionality, *Proc. of International Joint Conference on Fuzzy Systems and the 2nd International Fuzzy Engineering Symposium*, Yokohama, Japan, pp.2139-2146, 1995.
[34] K. Figueiredo, M. Vellasco, M. Pacheco and F. Souza, Reinforcement learning-hierarchical neuro-fuzzy politree model for control of autonomous agents, *The 4th International Conference on Hybrid Intelligent Systems*, Japan, pp.130-135, 2004.
[35] K. Figueiredo, M. Vellasco, M. Pacheco and F. Souza, Modified reinforcement learning – hierarchical neuro-fuzzy politree model for control of autonomous agents, *International Journal of Simulation Sys.*, vol.6, no.10-11, pp.4-13, 2005.
[36] K. Figueiredo, L. Campos, M. Vellasco and M. Pacheco, Reinforcement learning-hierarchical neuro-fuzzy politree model for autonomous agents – Evaluation in a multi-obstacle environment, *The 5th International Conference on Hybrid Intelligent Systems*, Rio de Janeiro, Brazil, pp.551-554, 2005.
[37] F. Martins, K. Figueiredo and M. Vellasco, Methods for acceleration of learning process of reinforcement learning neuro-fuzzy hierarchical politree model, *International Conference on Autonomous and Intelligent Systems*, 2010.
[38] L. P. Kaelbling, M. L. Littman and A. W. Moore, Reinforcement learning: A survey, *Journal of Artificial Intelligence Research*, vol.4, no.2, pp.237-285, 1996.
[39] M. Humphrys, *Action Selection Methods Using Reinforcement Learning*, Ph.D. Thesis, University of Cambridge, 1997.
[40] L. D. Pyatt and A. H. Howe, Decision tree function approximation in reinforcement learning, *Technical Reports*, Computer Science Department, Colorado State University, pp.98-112, 1998.
[41] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA, 1992.
[42] E. Ostergard, *Evolving Complex Robot Behaviour*, Master Thesis, University of Aarhus, Denmark, 2000.
[43] J. A. Boyan and A. W. Moore, Generalization in reinforcement learning: Safely approximating the value function, in *Advances in Neural Information Processing Systems*, G. Tesauro, D. S. Touretzky and T. K. Leen (eds.), Cambridge, The MIT Press, 1995.
[44] M. Bowling and M. Veloso, Rational and convergent learning in stochastic games, *Proc. of the 17th International Joint Conference on Artificial Intelligence*, pp.1021-1026, 2001.