A Hardware/Software Embedded Agent for Real-Time Control of Ambient-Intelligence Environments

Inés del Campo
Department of Electricity and Electronics
University of the Basque Country UPV/EHU
Leioa, Spain
e-mail: ines.delcampo@ehu.es

Koldo Basterretxea
Department of Electronics and Telecommunications
University of the Basque Country UPV/EHU
Bilbao, Spain
e-mail: koldo.basterretxea@ehu.es

Mª Victoria Martínez, and Javier Echanobe
Department of Electricity and Electronics
University of the Basque Country UPV/EHU
Leioa, Spain
e-mail: victoria.martinez@ehu.es

Faiyaz Doctor
Department of Computing and the Digital Environment
Coventry University
Coventry, United Kingdom
e-mail: aa9536@coventry.ac.uk

Abstract—This paper presents the development of an embedded intelligent agent able to perform real-time control of ambient-intelligence environments. The system has been implemented as a system-on-programmable chip (SoPC) on a field programmable gate array (FPGA). The scheme used for realizing the intelligent agent is an adaptive neuro-fuzzy system (NFS) enhanced with a principal component analysis (PCA) pre-processor. The PCA pre-processing stage allows a reduction of the input dimensions (features) with no meaningful loss of modeling capability. As a consequence, the computational complexity of the system is significantly reduced, allowing its implementation on a single electronic device. The NFS-PCA agent has been tested with data obtained in a real ubiquitous computing environment test bed. Results obtained show that the agent is able to perform real-time control of the environment in a proactive and non-intrusive way, and also to adapt to changes of user’s preferences in a life-long mode.

Keywords-component; ambient intelligence; field programmable gate array (FPGA); hardware/software co-design; intelligent environment; neuro-fuzzy system; principal component analysis (PCA); system-on-programmable chip (SoPC).

I. INTRODUCTION

The research activity concerning the multidisciplinary paradigm of Ambient Intelligence (AmI) has grown rapidly in the last decade [1]. AmI integrates the integration of intelligence and technology with the aim of supporting people in their everyday life. It refers to an environment, enriched with pervasive hardware (tiny processors, sensors, actuators, etc.), that is able to take decisions autonomously to benefit the environment users based on real-time information and historical data [2]. This idea can be deployed in a broad range of sectors such as housing, healthcare, mobility and transport, education, culture, leisure, etc [3]. Moreover, the development of AmI in large scale environments (e.g. public spaces or cities) is closer every day. In this sense, the Internet of Things –the Internet of smart devices and objects- will surely play an important role [4].

Several authors have proposed solutions based on intelligent agents and computational intelligence techniques, mainly fuzzy logic and neural networks, to add the intelligent component to different kinds of inhabited spaces in the above sectors. In [5] the authors proposed the use of intelligent agents based on fuzzy logic for realizing AmI in the Essex intelligent dormitory (iDorm). Along the same line, type 2 fuzzy agents for the iDorm have been reported in [6] and [7]. Another fuzzy agent with lifelong learning capability can be found in [8], the system integrates a speech recognizer with the agent and was tested in an intelligent classroom. Different agent-based approaches have been proposed to implement AmI-like strategies concerning energy and comfort management in buildings. A meaningful example is presented in [9] where several computational intelligence techniques have been combined –fuzzy systems, neural networks, and genetic algorithms– to develop an intelligent agent able to save energy while maintaining customer comfort levels in commercial buildings. In the same way, in [10] the use of neuro-fuzzy agents for the management of thermal and lighting comfort, indoor air quality, and energy conservation is discussed.

Concerning the technological dimension of AmI, in recent years a wide variety of hardware devices have been developed based on the advances produced in sensor devices, wireless communications, and Internet, for example, sensing sources such as surveillance cameras, smart indoor artifacts, wearable sensors, and mobile phones with embedded sensors (e.g. GPS receivers), among others [11]. A number of them integrate some kind of processing capacity –embedded intelligence– concerning their own functionality. However, a challenge still open in the consolidation of the ambient-intelligence paradigm is the implementation of the agent itself as a small hardware device, able to deal with the data gathered by the sensing devices, and capable of performing real-time control of the

This work was supported in part by the Spanish Ministry of Science and Innovation and European FEDER funds under Grant TEC2010-15388 and by the Basque Country Government under Grants IT419-10, S-PC10UN09, and S-PC11UN012.
environment in a stand-alone way or as a part of a multi-agent system. Actually, most of the intelligent agents for AmI applications have been implemented in software using general purpose computers, or microprocessors. In most cases, the complexity of the algorithms and the huge amount of data to be processed by them make an embedded hardware approach unsuitable. Hence, simplifying the complexity of the problem is a design objective for a single-chip intelligent agent.

A pioneering work in this regard can be found in [12] where the development of a single-chip embedded agent for AmI is proposed. The authors present a system-on-chip (SoC) approach of an adaptive neuro-fuzzy agent on a field programmable gate array (FPGA). To simplify the complexity of the problem, the authors proposed a feature selection technique (i.e., those inputs with less influence on the outputs are discarded). This solution has proven useful even in dealing with changes in the users’ habits, however, the selection of features involves a very time-consuming task, as is the exhaustive exploration of the solution space. For this reason the set of selected inputs was predetermined. To overcome this drawback, we propose a novel approach based on a feature extraction technique, the Principal Component Analysis (PCA), as a previous stage of the neuro-fuzzy system (NFS) computation. PCA is an approach to reducing the dimensionality by combining inputs linearly and identifying their priorities—non inputs are discarded.

This work presents the development of an adaptive embedded agent, based on a hybrid PCA-NFS scheme, able to perform true real-time control of AmI environments in the long term. The proposed architecture is a single-chip HW/SW architecture. It consists of a soft processor core (SW partition), a set of NFS cores (HW partition), the HW/SW interface, and input/output (I/O) peripherals. An application example based on data obtained in an ubiquitous computing environment has been successfully implemented using an FPGA of Xilinx’s Virtex 5 family [13].

The paper is organized as follows: Section II presents the proposed PCA-NFS scheme, and discusses key features of the model such as its approximation capability, and learning performance. Section III addresses the development of the HW/SW embedded system, and gives details of its FPGA-based implementation. Section IV provides experimental results obtained with the agent operating both in offline and online mode. Finally, Section V presents some conclusions.

II. INTELLIGENT AGENT MODEL

Recently, the combination of PCA with adaptive NFSs has been successfully applied to deal with large amounts of data in different areas (e.g., chemistry, biology, and medicine, among others). As will be seen, PCA preprocessing is also suitable for reducing the dimensionality of the dataset in AmI environments with no meaningful loss of the modeling capability and the learning performance of the NFS. The experimental data used in this work for the evaluation of the proposed PCA-NFS scheme were obtained at the Essex intelligent dormitory (iDorm); it is an ubiquitous computing environment test bed comprising a number of embedded sensors, actuators, processors, and a heterogeneous network [5]. The datasets were obtained during different living periods of the same user in the room. Seven input sensors were monitored during this phase: internal light level, external light level, internal temperature, external temperature, chair pressure, bed pressure, and time measured as a continuous input on an hourly scale. The controlled actuators were four variable intensity spot lights, the desk and bed side lamps, window blinds, and the heater. First, let us briefly introduce the main steps involved in PCA computation.

A. Feature extraction using principal component analysis

Principal component analysis is a linear orthogonal transformation used for reducing the dimensionality of data by omitting the components with lowest variances. Given a matrix \( \mathbf{X} \) of dimension \( m \times n \), where each row of the matrix is a vector \( \mathbf{x}_i = [x_{i1}, \ldots, x_{in}] \), \( i = 1, \ldots, m \), representing a single observation of \( n \) variables, the first step involved in PCA consists in rescaling the data. A new matrix \( \mathbf{U} \) of dimension \( m \times n \) is computed by subtracting the mean vector \( \mathbf{\bar{x}}_j \), \( j = 1, \ldots, n \), from each column of \( \mathbf{X} \) and dividing each element by the standard deviation \( \sigma_j \) of the corresponding variable.

\[
\mathbf{u}_i = \left[ \frac{x_{i1} - \bar{x}_1}{\sigma_1}, \ldots, \frac{x_{in} - \bar{x}_n}{\sigma_n} \right], \quad i = 1, \ldots, m \quad (1)
\]

Then, the covariance matrix, \( \mathbf{V} = \text{cov} (\mathbf{U}) \), is calculated and its eigenvectors and eigenvalues are computed. It is worth noting that the covariance matrix \( \mathbf{V} \) is an \( n \times n \) symmetric and real matrix, and thus also diagonalizable, \( \mathbf{D} = \mathbf{P}^{-1}\mathbf{V}\mathbf{P} \), so that the \( n \) column vectors \( \mathbf{p}_j \) of \( \mathbf{P} \) are the eigenvectors of \( \mathbf{V} \), and the diagonal entries of \( \mathbf{D} \) are the eigenvalues \( \lambda_j \). So, the data in \( \mathbf{U} \) can be approximately represented by selecting the \( p \) eigenvectors corresponding to the larger eigenvalues. The resulting data matrix \( \mathbf{Y} \) of dimension \( m \times p \) is calculated as follows:

\[
\mathbf{Y} = \left[ \mathbf{y}_{i,j} \right]_{i \times p} = \mathbf{U}_{\text{new}} \mathbf{P}_{\text{new}}^{T}, \quad p \leq n \quad (2)
\]

The selection of the principal components is done by considering a good percentage of variability (i.e., the sum of the variances of the \( n \) principal components). Let us perform a PCA analysis of the iDorm data in order to better understand the nature of these data. In the first place, the matrix of eigenvectors \( \mathbf{P} \) provides useful information:

\[
\begin{bmatrix}
-0.489 & -0.105 & -0.215 & 0.126 & -0.216 & -0.799 & 0.056 \\
-0.509 & -0.155 & -0.258 & 0.084 & -0.036 & 0.375 & -0.708 \\
-0.038 & 0.668 & -0.215 & 0.154 & 0.657 & -0.170 & -0.146 \\
-0.319 & 0.523 & -0.294 & 0.018 & -0.460 & 0.364 & 0.440 \\
-0.294 & 0.154 & 0.759 & 0.556 & -0.042 & 0.055 & -0.002 \\
0.389 & -0.173 & -0.409 & 0.798 & -0.102 & 0.050 & 0.034 \\
0.400 & 0.436 & 0.108 & -0.077 & -0.545 & -0.234 & -0.529
\end{bmatrix}
\]

Int. L.  Ext. L.  Int. T.  Ext. T.  Chair P.  Bed P.  Time
The above correlation coefficients show that the first component—first column vector—is strongly associated with light measures (internal light and external light/hour), whereas the second component is connected to temperature measures (internal temperature and external temperature), the third component is linked to the location of the user (chair pressure and bed pressure), etc. Although PCA does not consider the correlation between dependent variables and input variables (i.e. it is an unsupervised method), it can still be a good procedure for reducing input dimensionality because of the redundancy inherent in AmI data. Moreover, more than 80% of the variability of inputs is covered by the first three principal components (see Fig. 1). Therefore, the input space of iDorm could be reduced from seven to three or four dimensions (i.e. p=3 or p=4 in (2)) without hiding significant features.

Layer 1: every node in this layer is an adaptive node that evaluates the corresponding MF. For simplicity, triangular and normalized antecedents have been selected.

\[ O_k^{(3)} = \prod_{j=1}^{p} O_k^{(1)} = w_k, \text{ with } 1 \leq k \leq r. \] (4)

Layer 3: the node output of this adaptive layer is the sum of the weighted consequents,

\[ O^{(3)} = \sum_{k=1}^{r} O_k^{(3)} c_k = z \] (5)

Normally, an intermediate layer that performs the normalization of the rule activation has to be included, however, since we selected normalized antecedents, the sum of the weights of all the rules gives exactly one, so the normalization layer is not required.

The parameters of the above fuzzy system (i.e. antecedents and consequents) can be trained using the back-propagation gradient-descent method (GDM), or the hybrid learning rule proposed in [14]. This rule combines the GDM and the least-square estimator (LSE). In a forward pass, the consequent parameters are identified by the LSE method, while in a backward pass the antecedents are updated by the GDM.

B. Adaptive neuro-fuzzy inference system

As well as the fact that PCA is useful to extract the main features of the input space, neuro-fuzzy techniques can be applied to model the input/output mapping of the user behaviour. Neuro-fuzzy models have proven suitable for modeling large non-linear dynamic systems. Our approach to endow the agent with intelligence deals with a class of adaptive NFS based on the zero-order Takagi-Sugeno model. Consider a p-input single-output system with fuzzy rules:

\[ R_k: \text{IF } y_1 \text{ is } A_{p1}(y_1) \text{ and } y_2 \text{ is } A_{p2}(y_2) \text{ and } \ldots \text{ and } y_p \text{ is } A_{pp}(y_p) \text{ THEN } z \text{ is } c_k, \]

where \( R_k \) is the kth rule (1 \( \leq k \leq r \)), \( y_j \) (1 \( \leq j \leq p \)) are input variables, \( z \) is the output, \( c_k \) is a constant consequent, and the antecedents \( A_{pj}(y_j) \) are linguistic labels, each one being associated with a membership function (MF) \( \mu_{pj}(y_j) \). In a zero-order Takagi-Sugeno fuzzy model, the inference procedure used to derive the conclusion for a specific input \( y = [y'_1 \ldots y'_p] \) can be viewed as an adaptive network with the following layers (see Fig. 2):

\[ O_k^{(3)} = \sum_{k=1}^{r} O_k^{(3)} c_k = z \] (5)

![Figure 1. Variability (%) explained by the first five principal components. The thin line represents the accumulated variability. More than 80% of the total variability is explained by the first three principal components.](image1)

![Figure 2. Layered structure of the NFS model. The system is equivalent to a zero-order Sugeno-type inference model.](image2)
C. Hybrid PCA-NFS model

The above PCA feature extraction technique and the adaptive NFS model can be merged together to obtain a compact hybrid scheme with reduced complexity, good learning capability, and excellent modeling performance. This scheme will be used to develop an intelligent agent suitable for Ambient Intelligence environments. Figure 3 depicts an n-input s-output agent, where p principal components have been selected. Given a new observation in the input space arranged as a row vector, \( \mathbf{x} \), first a reduced representation of the input in the transformed space, \( \mathbf{y}^p \), is computed according to (1) and (2), and then the NFS network (3) to (5) is evaluated to infer each one of the system outputs.

![Real-time PCA-ANFIS control of the intelligent environment](image)

The PCA-NFS scheme has been successfully applied to the iDorm dataset. The procedure followed with the dataset to evaluate the model was similar to that used in previous works [5], and [12]. First, the 408 input/output vectors of the first dataset were randomized six times. Then, each one of the random sets was split into a training set and a validation/test set comprising 272 and 136 samples respectively. After that, the hybrid learning algorithm (i.e. LSE plus back-propagation GDM) was applied to adjust the parameters of the NFS. In this first round of experiments the four analog outputs (i.e. four intensity spot lights) were modeled taking into account all the available inputs (i.e. seven inputs) with the aim of computing achievable root mean squared error (RMSE) values for further comparisons. That is to say, all input features have been taken into account. In this case subtractive clustering was used to generate an initial NFS for each experiment. Table I (first column) shows the average of the best RMSE values obtained with the six randomized sets.

After that, the learning process was repeated, but with a previous PCA feature extraction step. That is, the inputs to the NFS were preprocessed by means of the PCA algorithm to discard the low variance components. Taking into account the percentages of variability shown in Fig.1, two alternatives were evaluated: a reduction of the dimensionality of the input space using four principal components (90 % of the variance) and three principal components (83.2 % of the variance). To be more realistic, the initial NFS configuration used in these experiments was a partition of the input universe with equally distributed antecedents –in fact, this will be the initial configuration implemented in the hardware agent. The same kind of hybrid learning algorithm LSE-GDM was applied. As can be seen in Table I, both rounds of experiments, four-input PCA and three-input PCA, provide RMSE values within the same magnitude order as the seven-input reference NFS. Therefore, it can be concluded that even three features could be enough to achieve acceptable errors in the context of this dataset.

<table>
<thead>
<tr>
<th>Output</th>
<th>Average of successful classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Four inputs</td>
</tr>
<tr>
<td>Window blind</td>
<td>92.10%</td>
</tr>
<tr>
<td>Bed light</td>
<td>95.62%</td>
</tr>
<tr>
<td>Desk light</td>
<td>94.68%</td>
</tr>
<tr>
<td>Heater position</td>
<td>81.55%</td>
</tr>
<tr>
<td>Overall</td>
<td>90.98%</td>
</tr>
</tbody>
</table>

D. PCA-NFS as a classifier

The above PCA-NFS system was slightly modified to cope with binary outputs. In the case of zero/one outputs, a step function was added to the output of the NFSs. Concerning the iDorm environment, the last considered outputs were treated in this way (window blind, bed light, desk light, and heater position). The averages of successful classifications are summarized in Table II. Note that in the case of binary outputs the PCA-NFS performs as a classifier, so the percentage of successful classifications is more relevant for evaluating the modeling performance of the system than the RMSE. It is noticeable that a two-input PCA-NFS provides acceptable classification percentages near to 90%, and a three-input PCA-NFS provides excellent results for the first three outputs –near to 99% of successful classifications in mean– and an acceptable value for the fourth output. Unlike the results obtained with analog outputs (see Table I), the modeling performance of a four-input classifier is not better than that of a three-input one (see Table II).

<table>
<thead>
<tr>
<th>Output</th>
<th>Average of successful classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Four inputs</td>
</tr>
<tr>
<td>Window blind</td>
<td>92.10%</td>
</tr>
<tr>
<td>Bed light</td>
<td>95.62%</td>
</tr>
<tr>
<td>Desk light</td>
<td>94.68%</td>
</tr>
<tr>
<td>Heater position</td>
<td>81.55%</td>
</tr>
<tr>
<td>Overall</td>
<td>90.98%</td>
</tr>
</tbody>
</table>
Taking into consideration the results obtained with the PCA-NFS scheme, applied to the iDorm data, it can be concluded that the dimensionality of the NFS can be significantly reduced by preprocessing the data with the PCA algorithm without sacrificing the modeling capability of the NFS. In terms of performance, the reduction of the dimension of the input space of the agent entails a reduction of the computational complexity of the problem, and therefore a reduction of hardware resources (device size), power consumption, and computation time.

III. ARCHITECTURE OF THE HW/SW AGENT

A milestone in the evolution of reconfigurable hardware has been to combine the logic blocks and interconnects of traditional FPGAs with embedded microprocessors, related peripherals, and memory blocks to form a system-on-a-programmable chip (SoPC). Some examples are the Virtex-5, and Virtex-6 families manufactured by Xilinx, which may include one or more PowerPCs embedded within the logic blocks. A similar approach, less efficient in terms of performance but more flexible, consists of using soft-processor cores instead of hard-cores that are implemented within the FPGA logic; two widely used soft-cores are Xilinx’s MicroBlaze, and Altera’s NIOS processors. These features of FPGAs have been exploited to develop heterogeneous HW/SW architectures in several application areas. A key feature for obtaining efficient HW/SW implementations is the partition of the system functionality into HW and SW blocks [15]. A widely accepted design rule states that to obtain a suitable flexibility/performance compromise, the regular and recurrent computations have to be implemented in the hardware partition while the irregular or less frequent computations are better suited to a software development. Next we will present the HW/SW architecture designed to implement the PCA-NFS agent. Without limiting the generality of the architecture, let us consider a seven-input eight-output agent. The agent will be used later to perform real-time experiments with the iDorm data, therefore, the input space will be reduced to three features (i.e. three principal components) according to previous discussions.

A. Hardware/Software partition

The system has been partitioned into six main modules with a well defined functionality: the I/O management, the change of basis (1) to (2), the NFS (feed-forward network) (3) to (5), the PCA computation (i.e. the computation of the eigenvectors of the covariance matrix), the LSE algorithm, and the GDM. Bearing in mind the above considerations concerning HW/SW partition, the first three modules are undoubtedly included in the category of regular/recurrent algorithms, while the latter three can be classified as irregular/less frequent ones for the present application. Moreover, the feed-forward network is the most time-consuming task involved in real-time control of the environment, so it is suitable to be located in the HW partition. On the other hand, the PCA computation and the learning algorithms –GDM and LSE– are not so critical because they will be activated at pre-defined times: a single time before the agent starts to control the environment (off-line computation and system set-up), and from time to time if changes in the user habits are detected (online adaptation). These algorithms, therefore, are best developed with a SW approach.

To get a better insight into the HW/SW partition problem, a C-language description of each one of the modules has been developed. Then, the timing performance of each C module -compiled on the target microprocessor- has been carefully profiled. Table III shows the results obtained using a typical clock frequency of 100 Mz. As can be seen, the change of basis takes only 13 μs, and the feed-forward network computation requires 25 μs. Note that the change of basis is performed only once per input measure, while the feed-forward network has to be evaluated several times -once per system output- that is actually a total of 200 μs for the case of the eight-output iDorm experiment.

<table>
<thead>
<tr>
<th>Time (ms) for a seven-input three-feature agent</th>
<th>Change of basis (3 features)</th>
<th>NFS computation (3 inputs)</th>
<th>PCA computation (7 inputs)</th>
<th>Offline training: GDM+LSE</th>
<th>Online adaptation: GDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change of basis</td>
<td>0.013</td>
<td>0.025</td>
<td>42</td>
<td>174</td>
<td>30.6</td>
</tr>
<tr>
<td>Time (ms) for a seven-input three-feature agent</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
</tr>
</tbody>
</table>

In view of the above considerations and the results provided in Table III, we developed the less frequent modules and the data pre-processing (i.e. I/O management and change of basis) in the SW partition, while the feed-forward networks (eight NFS cores) were implemented in HW (see Fig. 4).

![Figure 4. Block diagram of the HW/SW agent. The architecture has been developed around a MicroBlaze soft-processor core.](image-url)
Figure 4 depicts a system-level block diagram of the HW/SW agent. The system has been developed around a MicroBlaze microprocessor [16]. It is a 32-bit soft-processor core enhanced with a floating point unit (FPU) to accelerate the computation of the software modules. The architecture has been customized with several internal peripheral (I/O peripheral, UART, user switches, and a timer), 256 KB on-chip memory, and a controller for external SDRAM. The hardware partition consists of eight special-purpose NFS cores. Each core interfaces with the MicroBlaze by means of its own FSL (fast simplex link) bus. FSLs are high-speed point-to-point communication channels. The proposed solution was implemented using the XC5VSX50T device of Xilinx’s Virtex 5 family [13] - it is one of the smallest of this family. It has 8,160 Slices (each Slice contains four look-up tables (LUTs) and four flip-flops), 288 digital signal processing blocks (DSPs), 132 RAM memory blocks, and 6 phase locked loop (PLLs). Table IV summarizes the resources required to implement the seven-input eight-output agent.

<table>
<thead>
<tr>
<th>Resource type</th>
<th>Available resources</th>
<th>Consumed resources</th>
<th>Percentage of consumed resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice flip-flop</td>
<td>32.640</td>
<td>10.452</td>
<td>32 %</td>
</tr>
<tr>
<td>Slice LUT</td>
<td>32.640</td>
<td>11.367</td>
<td>34 %</td>
</tr>
<tr>
<td>DSP</td>
<td>288</td>
<td>222</td>
<td>77 %</td>
</tr>
<tr>
<td>36-kbit RAM block</td>
<td>132</td>
<td>99</td>
<td>75 %</td>
</tr>
<tr>
<td>PLL</td>
<td>6</td>
<td>1</td>
<td>16 %</td>
</tr>
</tbody>
</table>

B. Software partition

The software partition controls the behavior of the whole system. It has been developed in C language for the MicroBlaze processor. Figure 5 depicts a flow diagram of the steps involved in the agent operation. The user can interact with the agent in a straightforward way by means of general purpose input switches. When the user turns on the agent, the first step is the computation of PCA and the extraction of features according to a predefined percentage of variability. After that, the I/O samples are represented in the new basis according to (1) and (2). Then, the offline training step becomes active until all the outputs of the agent have been trained; training finishes when either a fixed number of iterations is performed, or a given approximation error is reached. The result of the offline training step is a set of parameters (antecedents and consequents) for each output. These parameters are then transferred to the hardware partition where they are stored in local RAM memories inside each NFS core. Once the agent is provided with a set-up model for each NFS core, the real-time control of the environment can be activated. As can be seen in Figure 5, the steps involved in real-time control are the acquisition of a new input vector, its transformation into the PCA basis, and the computation of the control action (i.e. activation of the NFS core in the HW partition). This three-step loop is repeated while the user feels comfortable with the agent behavior. Otherwise, the user is able to change the agent response manually. This change in the user preferences generates a new I/O sample. As long as the size of the training database does not change, whenever a new sample is inserted into the database, the oldest sample has to be removed. After a manual intervention of the user, the agent performs an online training iteration and updates the NFS cores involved in the changes (i.e. updates the RAM memories).

The calculation of the principal components has been implemented with the Jacobi algorithm [17]. This is the simplest method for the calculation of eigenvalues and eigenvectors, and it is recommended for matrices of moderate order. On the other hand, the LSE used in the offline training step to identify the consequent parameters, is accomplished by computing the Moore-Penrose pseudo-inverse [17]. This is an efficient method for inverting general matrix and usually the preferred way to solve a linear set of equations.

C. Hardware partition

The hardware partition consists of eight NFS cores, one per agent output. Each core communicates with the main processor.
by means of a pair of FSL links (send-receive). When the agent operates in the offline mode, the FSLs transfer the parameters of the trained models to RAM memory into each NFS core, while in the online mode the FSLs transfer the inputs of the agent, expressed in the new basis, to the inputs of the NFS cores and return to the microprocessor the outputs of the agent computed by the NFS cores (i.e. inferences).

![Architecture of the NFS cores.](image)

The main components of the NFS cores are the local RAM memory, the address generation unit (AGU), and the network data path. The RAM memory stores the antecedent and consequent parameters. These parameters are updated each time a new online adaptation step is performed. The AGU is mainly an arithmetic unit dedicated to compute the addresses in RAM of the parameters involved in the computation of the network for a given input vector. The network data path is a parallel arrangement of neurons, highly interconnected, that computes the three layers (3) to (5) of the NFS. The cores perform fuzzy inferences with a clock rate of 100 MHz. The first layer evaluates the antecedent membership function for a given input vector. This layer involves a sum and a product and requires only two clock cycles. The second layer calculates the activation of each rule, this involves a triple product per rule which lasts another two clock cycles. The third layer implements the aggregation of the rule activations (i.e. sum of products). It multiplies the outputs of the previous layer by the corresponding consequents, and sums them up. This layer involves four clock cycles. In sum, the 3-input cores require eight clock cycles to complete the inference computation, plus two additional cycles for HW/SW communications, that is, only 0.1 μs.

**IV. EXPERIMENTAL RESULTS**

The dataset from the iDorm has been used to perform real-time experiments with the HW/SW agent. Two sets of data were used, both corresponding to the same user but the measurements were taken in different months of the year. The first set of data (September) was used in the offline phase to develop a set-up model, while the second set (June) was used to perform real-time control and online adaptation. First, the offline procedure was performed for 100 iterations. Then, the agent was monitored in real-time for 200 additional iterations. Figure 7 shows the RMSE evolution for the four continuous outputs (variable intensity spot lights). As can be seen, the experimental results obtained in the offline phase agree with those obtained by simulation on the PC for the case of a three-input PCA (Table I). The transition between offline and online phases (iteration 101) shows a small RMSE increment, however, the agent is able to recover previous RMSE values, and even to improve them after several iterations. Concerning binary outputs (see Fig. 8), the average of successful classifications agree with those obtained by means of PC simulations in the offline phase (Table II). The online phase obtains classification percentages near 100% for three of the binary outputs (i.e. window blind, bed light, and desk light). The success of classification is rather poor in the case of the heater position for the first 100 online iterations, but it improves until a 100% success rate at around iteration 250.

The above experimental results validate the performance of the agent operating both in offline mode and in online mode. As can be seen, even when the agent is being subjected to strong online disturbances by injecting a new training pattern per iteration, it is able to adapt its parameters and retain the initial modeling performance –in practice new patterns will be considered only when the user disagrees with the agent behavior and decides to tune the actuators manually, so our experiments were developed assuming very stressful situations for the agent. It can be concluded that the synergy between PCA and the adaptive NFS performs in a very efficient way. Moreover, obtained results are similar to those reported in [12] where a feature selection technique was applied instead of the PCA feature extraction technique. These results are significant because PCA method is suitable for real-time computation, while the feature selection method requires an exhaustive exploration of the solution space, which is a very time-consuming task.

![RMSE curves for the analog outputs.](image)
V. CONCLUSION

Computational Intelligence techniques, mainly fuzzy logic and neural networks, are suitable for modeling intelligent agents in AmI environments. However, the computational complexity of these algorithms has to be reduced in order to deal with them in real time, mainly taking into account the large amount of information involved in typical AmI environments. The PCA feature extraction technique allows a remarkable reduction of the dimensionality of the problem without loss of meaningful information. In addition, it makes the agent more robust against failures in the environment sensors than other approaches such as the feature selection techniques. Moreover, as the idea behind PCA is to discard low variance components of the input data, it has the beneficial side-effect of filtering noise present in these data.

In this work we propose a hybrid PCA-NFS approach to develop intelligent agents which is simple enough to be implemented as a small embedded device while providing high performance and self-adaptation capability. Experimental results demonstrated the suitability of the HW/SW agent for performing real-time control of an AmI environment, even though the preferences and habits of the user changes over time. However, in the long term, if the agent is required to cope with more important changes in the user behavior, or a different user occupies the environment, a re-computation of the PCA basis could be required to avoid a reduction of the modeling performance of the agent. In this sense, in future works we are going to design new architectural features suitable for online computation of the PCA basis.

ACKNOWLEDGMENT

The authors would like to thank the researchers of the iDorm group for providing the data sets used to perform the experimentation.

REFERENCES