A Fault Tolerant Single-chip Intelligent Agent with Feature Extraction Capability

Koldo Basterretxea[a,*], María Victoria Martínez[b], Inés del Campo[b], and Javier Echanobe[b]

[a] *Department of Electronic Technology, Industrial Technical Engineering School of Bilbao, University of the Basque Country (UPV/EHU), Paseo Rafael Moreno 3, 48013, Bilbao, Basque Country, Spain*
[b] *Department of Electricity and Electronics, Faculty of Science and Technology, University of the Basque Country (UPV/EHU), Leioa, Barrio Sarriena s/n, 48940 Basque Country, Spain*

***Abstract*— Autonomy and adaptability are key features of intelligent agents. Many applications of intelligent agents, such as the control of ambient intelligence environments and autonomous intelligent robotic systems, require the processing of information coming in from many available sensors to produce adequate output responses in changing scenarios. Autonomy, in these cases, applies not only to the ability of the agent to produce correct outputs without human guidance, but also to its ubiquity and/or portability, low-power consumption and integrability. In this sense, an embedded electronic system implementation paradigm can be applied to the design of autonomous intelligent agents in order to satisfy the above mentioned characteristics. However, processing complex computational intelligence algorithms with tight delay constraints in resource-constrained and low power embedded systems is a challenging engineering problem. In this paper a single-chip intelligent agent based on a computationally efficient neuro-fuzzy information processing core is described. The system has been endowed with an information preprocessing module based on Principal Component Analysis (PCA) that permits a substantial reduction of the input space dimensionality with little loss of modeling capability. Moreover, the PCA module has been tested as a means to achieve deep adaptability in changing environment dynamics and to endow the agent with fault tolerance in the presence of sensor failures. For data driven trials and research, a data set obtained from an experimental intelligent inhabited environment has been used as a benchmark system.**

***Index Terms*— Autonomous intelligent agent; neuro-fuzzy systems; system-on-chip; fault tolerance; principal component analysis; ambient intelligence.**

1 Introduction

The term "intelligent agent" applies, in a broad sense, to a computer system that is capable of autonomous action in the environment it is placed in. To accomplish this task, the system must perceive its environment through sensors and must act upon it through effectors. Although there is not a universally accepted definition of what exactly an intelligent agent is, there are some properties that any intelligent agent is supposed to comprise, such as, autonomy, adaptability, reactivity, and proactivity. In particular, autonomy is always stressed as a key feature of intelligent agents, usually meaning the ability to act without the intervention of humans or other systems and to adapt their principles of behavior [1]. Being intelligent agents -at least in their initial phase of development- basically a paradigm for developing software applications, the concept of autonomy has traditionally been focused on algorithmic or functional independence from human guidance [2]. However, when intelligent agents are applied to domains which imply ubiquity (ambient intelligence, internet of things), or mobility (autonomous robots, spacecrafts etc) and concepts such as pervasive computing, power autonomy,

* Corresponding autor.
E-mail addresesses: koldo.basterretxea@ehu.es (K. Basterretxea), victoria.martinez@ehu.es (M. V. Martinez), ines.delcampo@ehu.es (I. del Campo), franciscojavier.echanove@ehu.es (J. Echanobe)

miniaturization, and mobile sensing are involved, the concept of autonomy must be applied not only to the *agent program* but also to the processing device or *agent architecture*.

The implementation of autonomous intelligent agents to produce invisible (hardware transparency), low-power, embedded systems able to incorporate the information coming in from multiple-sensor platforms and to "intelligently" process that information to produce outputs, involves many design challenges both in the computational paradigm to be applied and in the architectural or hardware structure to be created. Regarding this, the System-on-Chip (SoC) approach in which the processing cores, the memory and all interface blocks are integrated in a single chip, is a very appropriate solution to design and implement autonomous embedded intelligent agents [3], [4]. SoC implementations produce less power-hungry, smaller and more reliable systems compared to multichip implementations. An optimized SoC design, however, implies applying combined hardware and software design techniques, and requires a thorough knowledge of both the target technology and the intended application. These difficulties can be notably eased using FPGA (Field Programmable Gate Array) technology, which at the same time allows for high architectural adaptability and application versatility [5]. At the same time, FPGAs are moving to leading edge process technologies, and are being widely adopted in embedded system solutions because of their comparatively reduced cost in high-end embedded applications [6].

In this paper the design and implementation of a single-chip autonomous intelligent agent embedded in an FPGA is described. The agent intelligence is provided by a neuro-fuzzy computing core based on the PWM-FIS (Piece-Wise Multilinear-Fuzzy Inference System) architecture [7], which is a modification of the well known ANFIS (Adaptive Neuro-Fuzzy Inference System) architecture [8]. The applied modifications produce a more hardware friendly processing scheme, in order to obtain high performance neuro-fuzzy processors able to operate in real time (RT) with very short sampling times. Neural, fuzzy and hybrid neuro-fuzzy paradigms have already been adopted by some researchers to implement intelligent agents [9]-[20]. The neuro-fuzzy computational scheme embodies both main components of an intelligent agent: the *agent program* and the *agent architecture,* since knowledge is codified in its processing structure and in its parameterization. Self-adaptability and autonomy is achieved by means of learning algorithms that can be applied to optimize system performance in changing environments and to produce a system whose behavior is determined by its own experience (autonomy by learning from the environment). Yet there is an important drawback of neuro-fuzzy systems for their application to highly multidimensional problems when fast operation and/or reduced area and power is required, which is known as the *curse of dimensionality*. This phenomenon affects neuro-fuzzy processing in two ways: On the one hand, it makes learning processes much more ineffective and complex, slowing down system adaptation and sometimes impeding optimization algorithms from converging. On the other hand, if neuro-fuzzy high computing speed wants to be maintained by parallel processing architectures, it produces such an architectural size increase that it makes very costly or even impossible to embed such a system in a single chip.

To avoid the curse of dimensionality, increase learning and adaptation performances, and produce relatively small embeddable neuro-fuzzy intelligent agents, we have added a Principal Component Analysis (PCA) preprocessor to the agent architecture. PCA is a linear technique often used to identify patterns in data and, eventually, reduce data space dimensionality with little loss of information [21]. This is a very relevant technique to be adopted in many application areas of intelligent agents since very often these are multi-sensor scenarios in which thousands of data are collected from many system features or sensed variables. In reality, not always all sensed data are relevant to the agent operation

and very often these data are highly correlated, or their relevance may change depending on the evolution of the system/environment. In this sense the PCA processor is a feature extraction system that may also contribute to the interpretability of the neuro-fuzzy system operation. A PCA preprocessor can also be a good means for fault tolerance when communication from one or more sensors is lost. In this case, PCA can reorganize the input space to minimize loss of information and, eventually, improve the robustness of the system for fault tolerant operation.

To analyze how a single-chip neuro-fuzzy intelligent agent holding the properties described above can be designed and implemented, we used data from an ambient intelligence application which was carried out experimentally at the University of Essex by the Intelligent Inhabited Environment Group [11]. Researchers collected thousands of data in a monitored living space with an ubiquitous networked sensor system arrangement. The data were collected from the interaction with the environment of different users of this experimental "intelligent dormitory" during periods of several days in various seasons of the year. This is not a specially demanding application in terms of processing speed (at least in what refers to the input/output signal delay of the agent), but in any other aspect of the requirements of an embedded intelligent agent application (small size, low-power, multiple input/multiple output system, adaptability to a changing environment and autonomy) this is a very suitable real-world data set for the development of this research. In fact, trying to model and predict human activity and preferences over time, even for simple human-environment interactions, is an extremely challenging problem.

The paper is organized as follows: Section 2 is devoted to describing how PCA can be applied to the feature extraction and information rearrangement in the experimental data set. In Section 3 the proposed neuro-fuzzy processing scheme and its modification to operate with a PCA preprocessor that eventually reduces input space dimensionality is explained. The modeling capability of such a system is analyzed by performing some simulation-based experimental tests on the mentioned data-sets. The potential of PCA to obtain a more robust agent in the presence of sensor or communication failures is also investigated in this section. Section 4 is dedicated to a detailed description of the SoC design of the proposed system. Both the software partition and the hardware partition of the system are accurately explained and experimental performance and implementation figures are given. In Section 5 some selected simulation runs already presented in Section 3 are compared with the real operation of the agent implemented on an FPGA to show the actual performance of the real system. Finally some concluding remarks are given in Section 6.

2 Feature extraction by PCA preprocessing

As explained in the previous section, adding a PCA preprocessor to the input stage of the agent's architecture would modify the way acquired data is presented to the neuro-fuzzy information processing system. The object of this preprocessor is twofold. Firstly, to analyze the information provided by the multiple sensors distributed around the experimental environment and, eventually, to reduce the input space dimensionality with little loss of information. Secondly, to provide fault tolerance in case the communication with one or more of the sensors is lost or recovered

Principal component analysis is a linear non-parametric method for extracting relevant information from complex data sets [21][22], and it is a standard technique widely used for dimension reduction in statistical pattern recognition. Many experimental data sets are obtained from multiple sources (sensors), often with little or no previous knowledge of the underlying dynamics of the system being observed and modeled. In consequence, the collected information regarding some of the variables or presumed *features*, which form the basis of the input space of the system, may not be significantly

meaningful to describe system dynamics, and some of the selected variables may be highly correlated (information redundancy).

As mentioned in the introduction, the *Intelligent Inhabited Environment Group* from the University of Essex provided us with experimental data from some experiments carried out at their *Intelligent Dormitory* (iDorm). This test bed was designed for the research of intelligent agents in ambient intelligence environments, so that the agents can control them according to the needs and preferences of the user. As an illustrative example, we show the results obtained with two data-sets corresponding to the monitoring phases of the human interaction of one inhabitant with this environment during two different periods of time: the first one corresponds to four consecutive days in the month of March (Set1) while the second one corresponds to three consecutive days in the month of June (Set2). Data sets contain 7 input variables obtained from sensors distributed around the iDorm, and 8 outputs corresponding to as many effectors. Inputs and outputs are summarized in Table 1:

Table 1
Description of inputs and outputs in the experimental Intelligent Inhabited Environment.

Inputs	Variable	Outputs	Effector
In1	Internal light level	**Out1**	Dimmable spot light 1 (cont.)
In2	External light level	**Out2**	Dimmable spot light 2 (cont.)
In3	Internal ambient temperature	**Out3**	Dimmable spot light 3 (cont.)
In4	External ambient temperature	**Out4**	Dimmable spot light 4 (cont.)
In5	Pressure in chair (binary)	**Out5**	Blind state
In6	Pressure in bed (binary)	**Out6**	Bed light state
In7	Hour	**Out7**	Desk light state
		Out8	Heat state

Let us analyze the covariance matrix $\mathbf{C_X}$ corresponding to the normalized data-set after extracting the mean value:

$$\mathbf{C_X} = \begin{pmatrix} \mathbf{1402} & 1548 & -115.7 & 48.98 & -16.40 & -13.62 & 441.3 \\ 1548 & \mathbf{1709} & -127.5 & 54.24 & -18.19 & -15.03 & 486.95 \\ -115.7 & -127.5 & \mathbf{10.47} & -3.527 & 1.479 & 1.085 & -36.02 \\ 48.98 & 54.24 & -3.527 & \mathbf{2.698} & -0.242 & -0.778 & 15.09 \\ -16.40 & -18.19 & 1.479 & -0.242 & \mathbf{1.201} & -0.428 & -5.738 \\ -13.62 & -15.03 & 1.085 & -0.778 & -0.428 & \mathbf{1.124} & -3.826 \\ 441.3 & 486.95 & -36.02 & 15.09 & -5.738 & -3.826 & \mathbf{140.3} \end{pmatrix} \quad (1)$$

From this matrix useful information can be elicited: variability of data is very strong in the first two input vectors (In1 and In2), and also in vector In7, but very weak in vectors In3, In4, In5 and In6. And, what is more interesting, correlation between variables can be clearly observed: correlations are very strong between vectors In1 and In2, and also between In1 and In7, and In2 and In7. Correlations between In1-In2 and In3 are also notable. It is quite clear, therefore, that this input space basis does not describe properly data variability and that redundancy is high. Now we apply PCA analysis to these data and obtain the transformation matrix **P**.

$$\mathbf{P} = \begin{pmatrix} \mathbf{0.444} & \mathbf{0.491} & -0.134 & 0.242 & 0.377 & -0.343 & \mathbf{0.475} \\ -0.234 & -0.116 & \mathbf{0.687} & \mathbf{0.556} & 0.270 & -0.243 & 0.140 \\ 0.020 & -0.333 & -0.174 & -0.430 & \mathbf{0.469} & \mathbf{-0.613} & 0.280 \\ 0.422 & -0.091 & 0.176 & -0.126 & 0.587 & 0.617 & 0.206 \\ -0.624 & 0.161 & 0.159 & -0.382 & 0.338 & 0.178 & -0.518 \\ -0.324 & -0.268 & -0.639 & 0.535 & 0.307 & 0.188 & -0.019 \\ 0.275 & -0.727 & 0.128 & 0.007 & -0.115 & -0.019 & -0.605 \end{pmatrix} \quad (2)$$

PCA outputs do not necessarily contain interpretable relationships with original data, but looking to the values of the principal coefficients in (2), and recalling that these coefficients express the correlation between the principal components and the original base components, the information contained in the most relevant principal components seems quite obvious in this case. The first component is clearly related to ambient light measures (and to time which, as pointed out above, is tightly correlated with the former), the second component is linked to ambient temperature measures, and the third component seems to contain information about location of the user (chair pressure sensor and bed pressure sensor). This is particularly interesting for this intelligent agent design since, as we will see, its fuzzy processing scheme enables further interpretability of system operation.

Once the basis rotation performed by PCA is done, how experimental data is expressed in the new basis can be analyzed. Let us inspect the percentage of total data variability that is expressed by each of the principal components:

$$variance(\%) = \begin{Bmatrix} 47.629 & 24.095 & 12.167 & 6.358 & 5.114 & 3.327 & 1.309 \end{Bmatrix} \quad (3)$$

According to (3), more than 70% of data variability is covered for Set1 with only two principal components, more than 80% is covered with three components, and more than 90% with four components. Hence, it is clear that applying input vector space reduction is feasible and sensible in this case, and that this technique may dramatically reduce curse of dimensionality issues in the neuro-fuzzy intelligent agent.

3 The Intelligent Agent: Information processing and adaptability

3.1 The PWM_FIS processing cores

Although a detailed description of the PWM-FIS computational scheme can be found in [7], it is convenient to briefly describe this neuro-fuzzy processing system developed by the authors, which is based on the ANFIS inference paradigm [8] and which constitutes the core processor of the proposed intelligent agent. The Gaussian membership function-based ANFIS inference scheme is suitable for parallel hardware implementation by providing a data path for each rule and membership function (MF) circuits for each antecedent. This parallel configuration would provide fast operation but is very resource-consuming. A set of restrictions specially suited to digital hardware implementations is based on selecting an input space partition which consists of normalized triangular MFs with single overlapping in each input dimension. On the one hand, normalized membership functions verify $\sum_j w_j = 1$; this avoids the time-consuming division operation in the output, where the average operator can be replaced by a sum operator. On the other hand, by limiting the overlapping of the antecedents to two, only two antecedents per input dimension provide non-zero membership values. Moreover, the pair of active antecedents provides complementary membership values (i.e. the sum of both

membership values gives one), which further simplifies the computation of the activation strength of the rules.

The above described constraints ensure that given an input vector $\boldsymbol{x} = (x_1, x_2,\ldots,x_m)$ the pairs of active antecedents define an "active cell" in the input domain. Once this active cell is identified and its corresponding parameters are loaded, a single inference kernel can be used to process the active rules. Since the input vector activates only two antecedents per input, the number of active rules at each time is reduced to 2^m, and the ANFIS inference algorithm can be rewritten as follows in four levels of computation:

Level 1: Membership function evaluation.

$$\mu_{ij}(x_i) = a_{ij}(x_i - b_{ij}), \quad 1 \le i \le m, \quad 1 \le j \le 2^m \tag{4}$$

Level 2: Rule activation.

$$w_j = \prod_{i=1}^{m} \mu_{ij} \quad \text{with} \quad 1 \le j \le 2^m \tag{5}$$

Level 3: Weighted activation.

$$P_j = w_j c_j \quad , \quad 1 \le j \le 2^m \tag{6}$$

Level 4: Output level.

$$y = \sum_{j=1}^{2^m} P_j \tag{7}$$

The above inference algorithm (4) to (7) provides a piecewise-multilinear (PWM) output, so this computing algorithm was termed the PWM-FIS. The main advantages of the proposed constraints are a reduction of the computational cost of the algorithm and a reduction of the arithmetic complexity of operators. Moreover, the above described PWM inference model can be viewed as a four layer neural network so, like the ANFIS, it can be trained to approximate arbitrary mapping surfaces by means of parametric optimization algorithms.

3.2 The PCA preprocessor

A PCA preprocessor has been added to the input stage of the above described PWM-FIS processing scheme. As mentioned above, the PCA preprocessor is aimed at analyzing the incoming information from the environment sensors and rebuilding the input space of the system in order to extract the relevant features of the dynamics under control and, eventually, reduce input space dimensionality. This information preprocessing helps the neuro-fuzzy processor to enhance its modeling performance while avoiding curse of dimensionality associated problems and simplifying the computational load at learning stages. A processing scheme of the PWM-FIS with PCA preprocessor is depicted in Fig. 1.

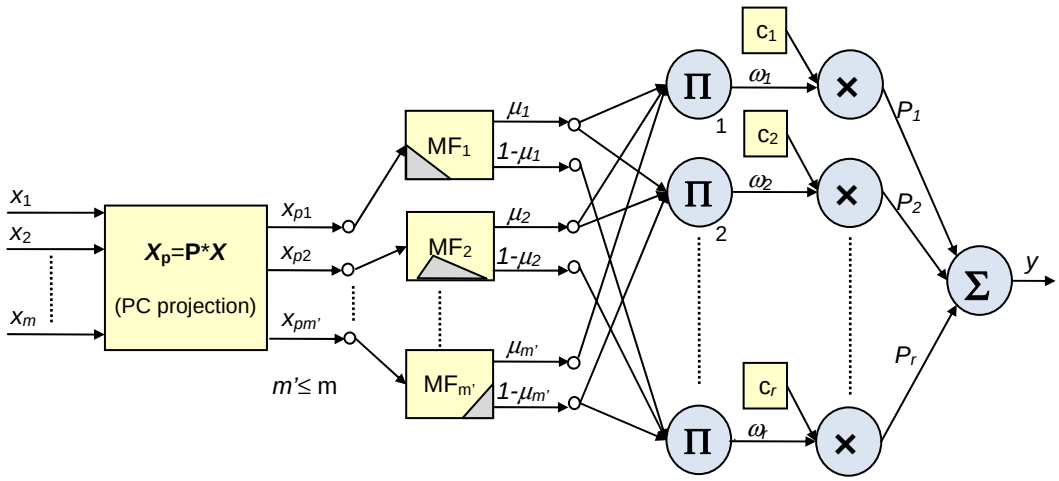

Fig. 1: Information processing scheme of the modified PWM-FIS core. The input space dimensionality is eventually reduced after the PCA preprocessor.

The modified PWM-FIS cores (the new base projector plus the PWM-FIS processing net) together with a finite state machine that will provide the appropriate control signals for the operation of the agent, are the basis structure of the intelligent agent whose SoC implementation is described in Section 4 (see Fig. 2). We will refer to this information processing structure as the Information Featuring Adaptive Neuro-Fuzzy Agent (IFANF Agent).

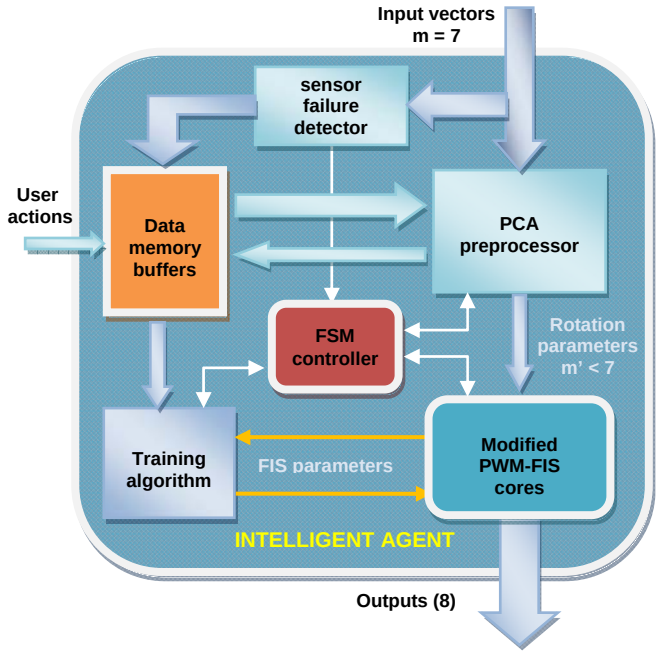

Fig. 2: Simplified operational scheme of the IFANF agent.

3.3 Modeling power and adaptability of the IFANF agent in a reduced vector space

To test the modeling power of the analyzed experimental data-sets by the IFANF agent, the system has firstly been software programmed and its operation simulated with machine precision using a PC and Matlab programming tools. A virtual experiment was designed with two main objectives: The first objective was to test the modeling capability of the proposed intelligent agent using reduced vector spaces and to analyze how applying

PCA during agent on-line operation could help to improve system performance in the presence of changes in the underlying dynamics. The second objective was to investigate whether an on-line PCA preprocessor can be used to provide the agent with fault tolerance if the communication link to one or several of the sensors is lost.

The experiments were set up as follows: SET1, which contains 440 samples collected during four days in the month of March, is split into two subsets containing 280 samples (Set1.1) and 160 samples (Set1.2) respectively. Set1.1, which in a real system would be obtained after an initial monitoring period of the user behavior, is used as the training set for the initial off-line training round. Set1.2 is merged with Set2, which contains 585 samples collected during three days in the month of June (745 vectors altogether). We label this set as Set3 (see Fig.3), which is used to emulate the system operation in real time (RT-mode operation). In the system initialization stage the agent loads Set1.1 and splits it randomly into a training set containing 200 vectors and a validation set containing the remaining 80 vectors. PCA is applied to this set and a new principal component basis input vector space is obtained. Vector space dimension is reduced according to a previously selected reduction criterion (usually a desired minimum percentage of data variance representation). Applied training algorithm is a hybrid gradient descent plus LSE (Least Square Estimator) [8], and the PWM-FIS core is parameterized with the parameters that minimize the validation set modeling error. The number of MF's for each input space vector was set to three in all experiments (see [24] for further explanations on this matter).

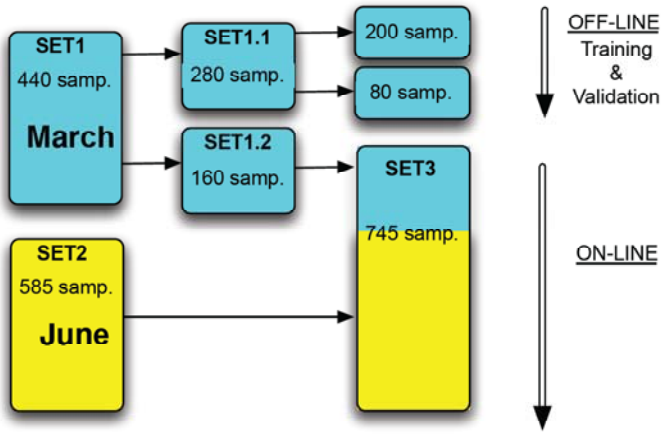

Fig. 3: Rearrangement of experimental data for off-line learning and on-line adaptation

After this off-line learning stage, the system should have captured the preferences of the user to a sufficiently satisfactory level, so it then switches to *RT-mode* (Real Time mode or on-line mode) and takes control of the actuators that modify the environment conditions. Incoming new vectors are projected into the reduced vector space before entering to the FIS cores. The agent, however, will continue learning and adapting itself on-line during RT operation. RT operation is simulated by feeding vectors from Set3 one by one at every iterations and applying a pattern-by-pattern on-line version of the same training algorithm: at each "sampling instant" a new vector is added to a 200 vector wide learning matrix from which the oldest vector is removed. The agent can be programmed to leave RT-mode and switch to off-line mode to perform a new PCA and new off-line training if certain circumstances concur: modeling error is growing during a previously set maximum time span, modeling error grows over a previously set threshold, communication link with a sensor is lost or recovered, or simply if certain time period has been reached. Obtained results at several programmed scenarios are summed up below for one of the outputs of the system (Out1: Dimmable spot light 1):

Experiment-1: Minimum covered data variability after PCA is set to 80%. No return to off-line mode is programmed.

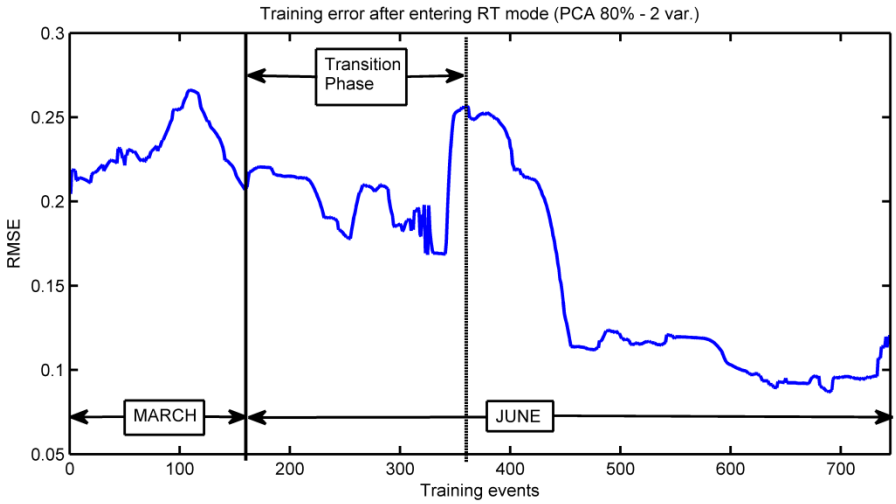

Fig. 4: Modeling error for 80% of data variability represented by principal components (2 input variables).

As depicted in Fig. 4, the evolution of the modeling error during RT-mode operation is mediocre for the data-set corresponding to March and for the "transition phase" (the transition phase refers to the time span during which the training matrix, containing 200 vectors, is being emptied of data from the March data-set and filled with data from the June data-set). After the transition phase the modeling error is lowered to much better performance values ($RMSE \approx 10\%$), since the behavior pattern reflected by this last data-set is easier to model. However, it must be remarked that the reduced input space produced after off-line PCA comprises only two principal components in this particular experiment, so the obtained modeling error is not so big considering such an extreme simplification.

Experiment-2: Minimum covered data variability is set to 90%. No return to off-line mode is programmed.

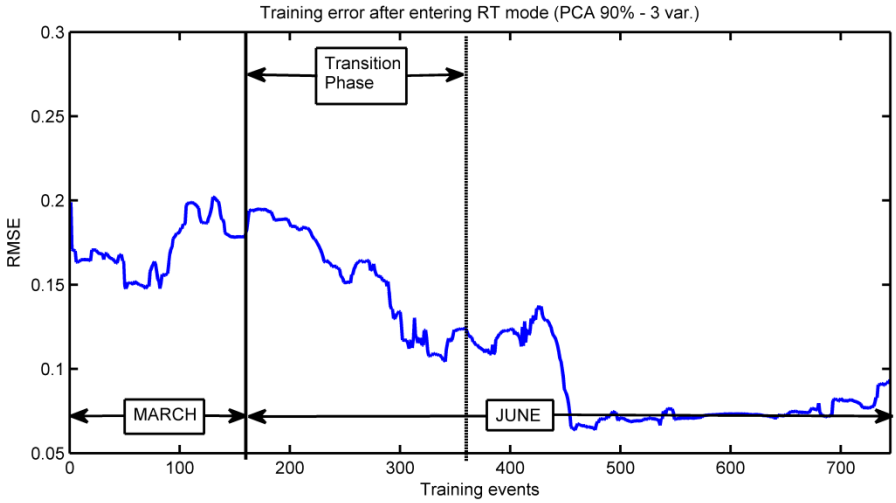

Fig. 5 Modeling error for 90% of data variability represented by principal components (3 input variables).

In this experiment, obtained reduced input space dimensionality after PCA is three, so the obtained general training error is better compared to Experiment-1, as can be clearly seen in Fig. 5. This improved modeling capability is, of course, achieved at the cost of a more

complex computing scheme, but still much simplified compared to what would be necessary to deal with a seven dimension input space.

Experiment-3: This experiment initiates the run dedicated to test the suitability of PCA to provide fault tolerance when one or more signals from the sensors providing environmental information are lost. We set the minimum covered data variance after PCA to 90% as in Experiment-2, but now two cumulative sensor failures were programmed. To evaluate the potential of PCA for compensating this loss of information, we programmed two different agent operation controllers: In the first one, the FSM produces an interruption and goes back to the off-line training mode to perform new off-line adaptation with the vectors stored in the data memory (of course removing the column corresponding to the missed sensor), but it does not perform a new PCA. In the second version a new PCA is performed with the matrix stored in memory previous to off-line training. In this experiment the failures were programmed in sensors 1 and 2 consecutively which, as reflected by (2), are the most relevant providers of information for the most important principal component (containing the information regarding to ambient illumination). A comparative graph of the modeling error evolution is depicted in Fig. 6.

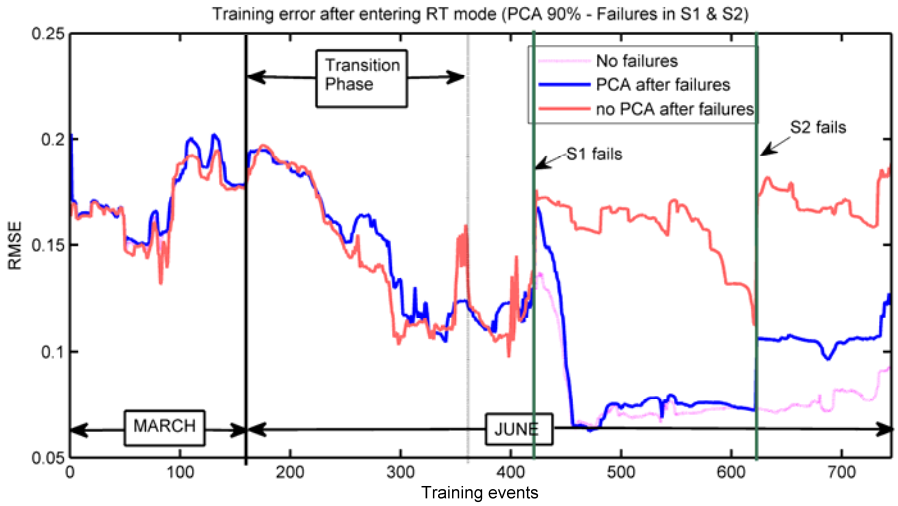

Fig. 6. Modeling error with 90% of data variability represented by principal components and two consecutive sensor failures (S1 & S2). Comparison of system performance when a) there are no sensor communication failures, b) PCA is performed after a failure is detected, and c) no PCA is performed after a failure is detected.

In this graph it can be clearly seen how system learning is perturbed when a sensor communication failure event occurs. When the system switches to off-line training mode to readapt its parameters but no PCA is performed, it is unable to follow the reference modeling RMSE line corresponding to the behavior of the system with no sensor failures (dashed line in Fig.6), and a sudden error growth is observed. But when a new PCA is performed after a sensor failure, the computation of the new base based on the remaining sensor information produces a remarkable enhancement of the modeling power of the system, with no noticeable loss of ccuracy until the second sensor is switched off.

Experiment-4: In this experiment the programmed sensor disconnections were sensors 3 and 4, corresponding to ambient temperature measures, and which contain the most meaningful information for the second principal component (see (2)). Fig. 7 depicts a comparative modeling error evolution for this scenario. Once again, PCA shows its potential to compensate for the information loss produced by sensor communication failures.

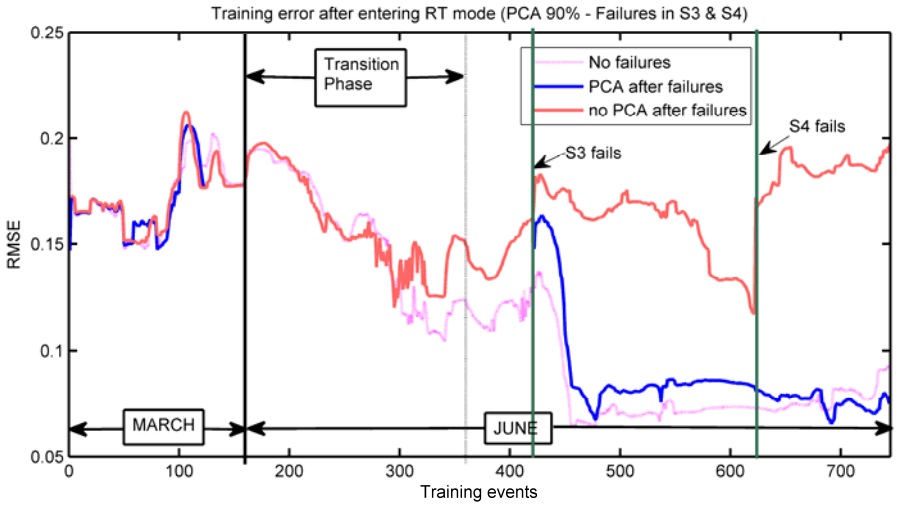

Fig. 7. Modeling error with 90% of data variability represented by principal components and two consecutive sensor failures (S3 & S4). Comparison of system performance when a) there are no sensor communication failures, b) PCA is performed after a failure is detected, and c) no PCA is performed after a failure is detected.

Experiment-5: Failures were consecutively programmed on sensors 5 and 6, which contain the information regarding user position (bed and chair pressures). Results are depicted in Fig. 8. The loss of the information provided by the bed pressure sensor is successfully compensated by the PCA. However, the cumulative failure of both position sensors seems to be too restricting for the agent's operation.

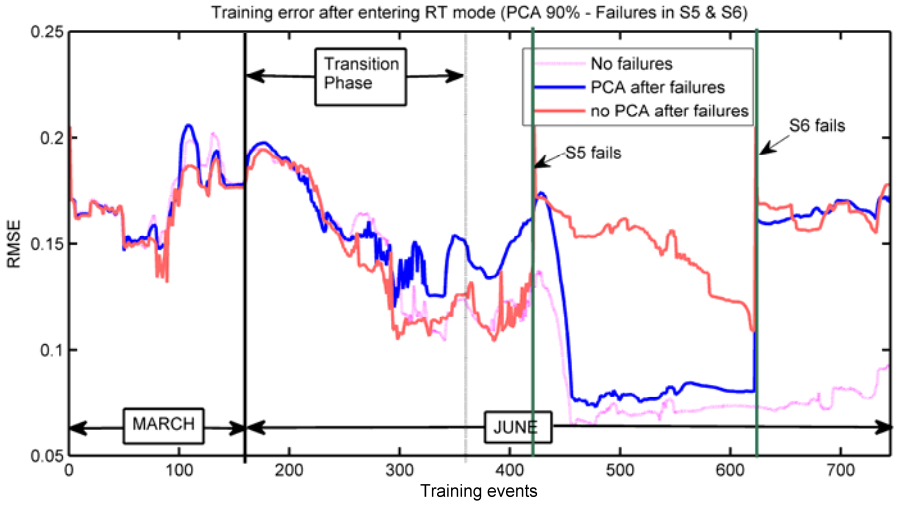

Fig. 8. Modeling error with 90% of data variability represented by principal components and two consecutive sensor failures (S5 & S6). Comparison of system performance when a) there are no sensor communication failures, b) PCA is performed after a failure is detected, c) no PCA is performed after a failure is detected.

Experiments-6 and -7: In these experiments we explored other combinations of sensor failures combining information losses from ambient light and ambient temperature, and the loss of the time information (clock) with ambient light information. Results are depicted in Figs. 9 and 10. As can be observed, the system is very sensitive to the loss of information from the clock signal (see Fig. 10), which is reasonable since a correlation analysis between the output and this variable reveals a strong dependence. However, the PCA is able to keep error at acceptable levels.

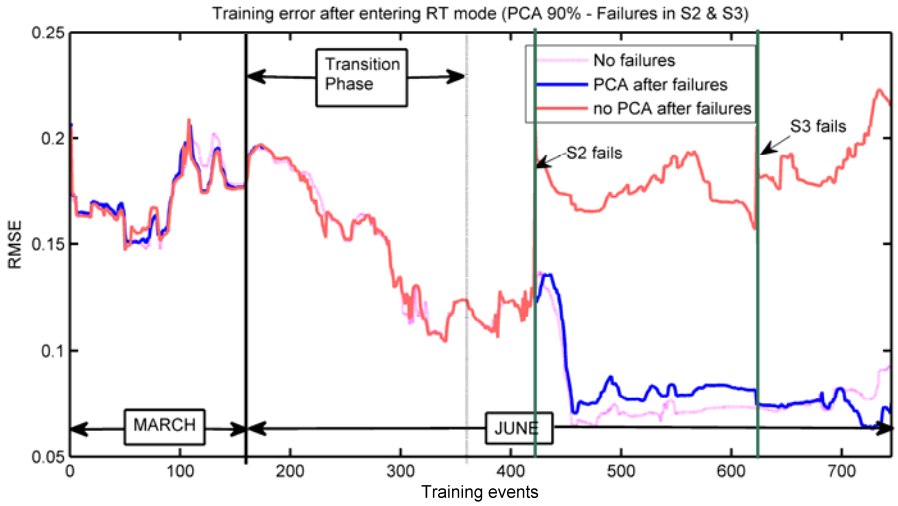

Fig. 9. Modeling error with 90% of data variability represented by principal components and two consecutive sensor failures (S2 & S3). Comparison of system performance when a) there are no sensor communication failures, b) PCA is performed after a failure is detected, c) no PCA is performed after a failure is detected.

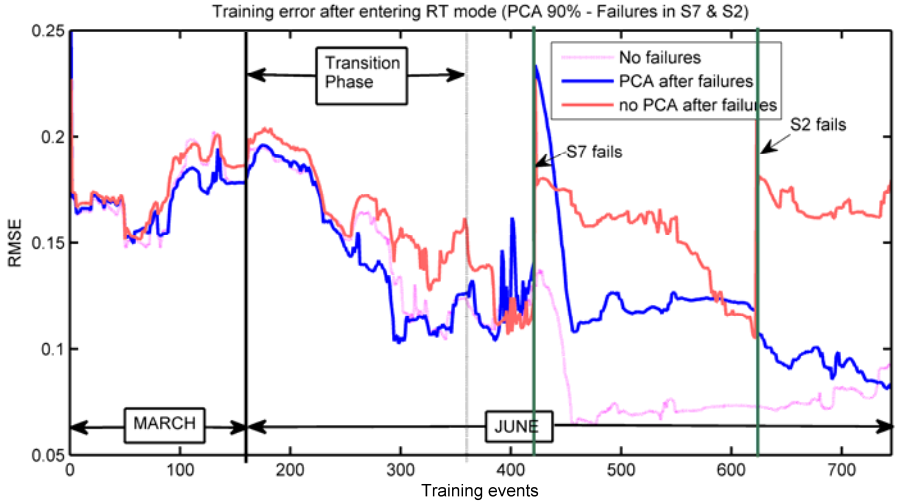

Fig. 10. Modeling error with 90% of data variability represented by principal components and two consecutive sensor failures (S7 & S2). Comparison of system performance when a) there are no sensor communication failures, b) PCA is performed after a failure is detected, c) no PCA is performed after a failure is detected.

The same set of experiments has been run with the experimental data obtained from the rest of the outputs (Out2 to Out8) corresponding to this user, and for similar data-sets corresponding to other users of the *iDorm* environment. Very similar modeling results have been obtained and PCA has shown its suitability to provide deep adaptability and fault tolerance in all cases.

3.4 Benefits of a reduced input space dimensionality

Reducing highly dimensional input spaces is extremely important for neuro-fuzzy processing, particularly when an embedded implementation is the target. As mentioned above, neural/fuzzy systems are adaptive parallel processing systems that suffer from the so called *curse of dimensionality*. That means that the cost of both implementing the computational system and computing an output increases exponentially fast as the input space dimension grows. Moreover, learning/adaptation processes are much heavier in highly dimensional spaces and usually give rise to systems with poor generalization abilities. Besides, system interpretability is also reduced. This is particularly true for many application areas of intelligent agents, such as the experimental test-bed for ambient

intelligence applications analyzed in this work, where the information which is relevant for modeling the system and performing predictions is very often "hidden" among the many available input variables. As we have seen above, PCA can be a powerful tool for extracting the relevant information and producing notably simpler input spaces that still provide the required information for a good system performance. Unlike feature selection techniques, which require an extensive search for the optimal combination of input variables that takes a lot of computational time, an embedded PCA processor allows the system to extract updated environment features while it is running when signs of performance degradation are detected. PCA takes comparatively little time and performed space rotation usually produces comparatively smaller dimensionalities.

We can now more precisely analyze how the intelligent agent under analysis benefits from a reduced input vector space, dramatically reducing the number of rules and parameters compared to a system that would be operating with the original variables. With that aim, let us analyze the PWM-FIS processing scheme defined in (4)-(7). Letting m' be the number of inputs or variables, and p the number of membership functions per input, each PWM-FIS core evaluates $r = 2^{m'}$ out of a total $p^{m'}$ fuzzy rules and has to adapt $(p-2)m'+p^{m'}$ parameters at each training epoch. It performs $2^{m'}+m'-1$ sums/subtractions and $m'(2^{m'}+1)$ product operations to produce an output for each input. So, considering that in this particular application a good system performance has been observed with 2 to 4 input variables, obtainable complexity reduction is summed up in Table 2. More explicit figures will be presented in the following sections when describing the actual agent architecture implemented as a SoC on the FPGA.

Table 2
PWM-FIS agent complexity comparison for a three membership functions per input system.

PWM-FIS core	7 var. (original space)	4 var. (PCA)	3 var. (PCA)	2 var. (PCA)
Total rules	2187	81	27	9
Computed rules	128	16	8	4
Parameters	2194	85	30	11
Adds/subs	134	19	10	5
Mults	903	68	27	10

4 The single-chip IFANF agent

Embedded systems are tightly constrained systems developed to perform a single functionality. Small size, low cost, reduced power consumption, and high processing speed for real time operation are usually their main requirements. To achieve these attributes the internal architecture of many embedded systems consists of hardware and software partitions in a single chip. In fact, the coexistence of both hardware and software provides them with enough flexibility without sacrificing performance requirements. The architectural and technological features of FPGAs make them especially suited to developing single-chip embedded systems. Current FPGA families combine logic blocks and interconnects, typical of traditional reconfigurable devices, with embedded microprocessors, peripherals, and memory blocks to form a SoPC. We have exploited all these resources to develop an embedded single-chip IFANF agent on an FPGA.

The system level architecture of the proposed approach is shown in Fig. 11. It is a HW/SW architecture developed around a *MicroBlaze* soft-processor [23]. The system consists of the processor –with its associated peripherals and buses–, a set of custom logic cores that compute the PWM-FIS algorithm in parallel (i.e. PWM-FIS cores), and the Fast-simples-link (FSL) bus system that communicates the *MicroBlaze* (SW partition) with the PWM-FIS cores (HW partition). The SW partition implements the system controller and several specific tasks, mainly, I/O processing, PCA computation, sensor failure management, and learning/adaptation (both off-line and on-line learning), while the HW partition performs high-speed fuzzy inferences for real-time operation of the agent. The

microprocessor has been configured with advanced features (i.e., floating-point unit (FPU), hardware multipliers, and data cache) to meet the performance requirements and the precision requirements of both the learning algorithms and the PCA computation.

4.1 Software Partition

The software partition, which runs on the *MicroBlaze* embedded processor, controls the overall operation of the system: I/O processing, PCA computation and vector projection, sensor failure management, and learning/adaptation tasks. A simplified diagram of the FSM in charge of the upper level system control is depicted in Fig. 12, showing the main transitions and tasks. When the system is started, it enters state S0. This is an off-line mode operation state where operation conditions are initialized and a new agent model is built. First, the sampled data-set containing a collection of I/O observations stored during the previous monitoring phase is loaded. A PCA computation is then accomplished on the data-set, determining the dimensionality reduction of the input space and setting the corresponding vector projection operator that will transform every new incoming data in real-time mode operation. Data in the original data-set are transformed likewise, thus producing the modified training set which is then used in the off-line learning stage to generate the initial model. Once the learning phase is concluded, the processor initiates the transference of the model parameters to the PWM-FIS cores.

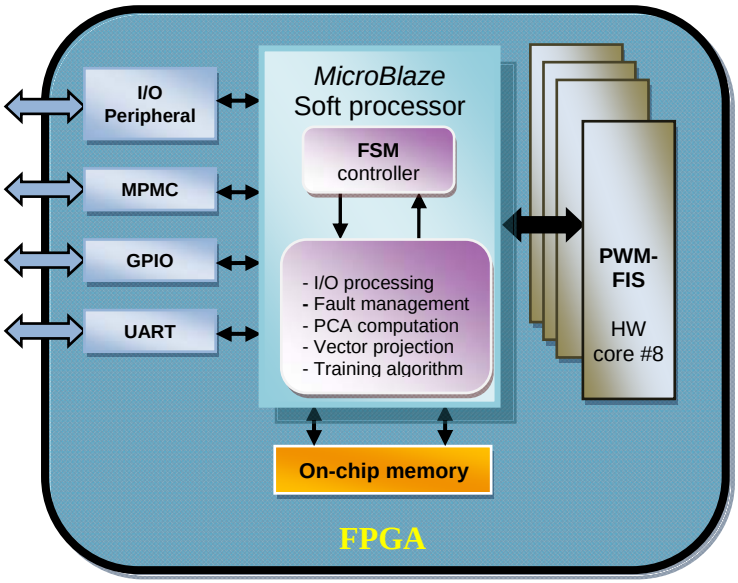

Fig. 11: System level architecture of the IFANF agent embedded in an FPGA. An embedded soft processor runs the SW partition while the HW cores provide parallelism and high processing speed for the PWM-FIS calculations.

When the agent model is available, the machine moves straightaway to state S1. This state is responsible for the real-time control of the environment while in normal operating conditions. The system remains in the state S1, awaiting incoming data. On receiving a new information vector from the sensor network, the data projection is carried out, and the reduced vector is then transferred to the FIS cores to produce the output. While in state S1, the user might manually change the controls in the intelligent environment. The system reacts to the user intervention by storing the new so generated I/O vector in the sample set, from which the oldest observation is removed. Then a one-step on-line learning stage is performed on the resulting training set to readjust the system model.

In real-time mode, the system is able to manage sensor failures. When a failure or absence of communication is detected in any input the system moves to state S2, and activates a recovery procedure where the model parameters are recalculated. This

implies returning temporarily to an off-set mode to compute a new PCA transformation on the current sample set, from which the stored data corresponding to the faulty sensor are discarded for calculation. Again, after a new learning stage, the updated model parameters are transferred to the PWM-FIS processing hardware cores. For each new sensor failure or restored sensor communication an equivalent series of actions is carried out to adjust the model. While in state S2, the machine performs all real-time control tasks, as in state S1, although every incoming data is previously handled according to actual sensor conditions. The system therefore remains in state S2, and the agent operative, until all of the communication channels with the environment sensors are restored. Then the system returns to the normal operation state S1.

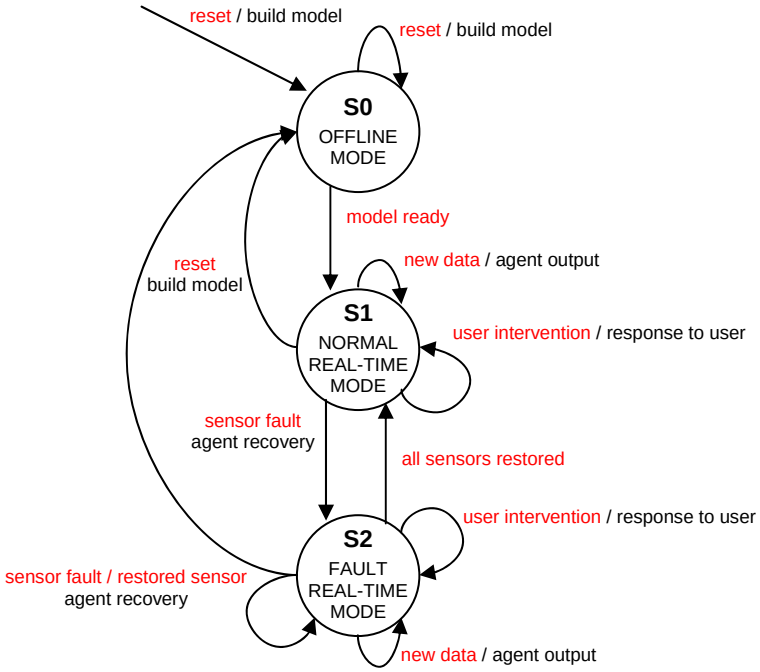

Fig. 12. Finite State Machine controller of the IFANF agent

4.1.1 The Learning Algorithm

An embedded software learning algorithm has been programmed to be run in the *MicroBlaze* soft processor. The hybrid LSE + SD (Least Square Estimator + Steepest Descent) algorithm used to train the PWM-ANFIS has been implemented in the form of a single step LSE pass (forward stage) followed by multiple SD passes [8]. In the off-line learning stage (S0 state), a 100-iteration batch algorithm is applied while in the on-line or RT stages (S1 and S2 states) a pattern-by-pattern version of the algorithm updates system parameters in just one iteration. Since the *MicroBlaze* floating point unit (FPU) uses the IEEE 754 standard single precision format, the possible numerical issues that could come up after on-chip implementation of the IFANF agent were carefully studied by simulating the operation of a custom learning C program with single precision arithmetic. As expected, the updating of the nonlinear parameters by the SD algorithm did not produce noticeable deviations compared to the machine precision runs computed on a PC. On the contrary, running a single precision LSE algorithm on data produced by this application is much more delicate.

The LSE is applied to solving the linear system that comes up when updating the linear parameters of the PWM-FIS cores (consequents of the Sugeno-type FIS):

$$\mathbf{A}\theta = y \tag{8}$$

where θ is the unknown vector containing the linear parameters to be updated and **A** is the involved matrix. The size of **A** is determined by the amount of input vectors stored for each learning stage and by the quantity of consequent parameters, which in turn is determined by the number of input variables. In the performed experiments, the amount of vectors for training was 200, and the number of inputs oscillates between 2 and 4, depending on the results obtained from the PCA preprocessor. That means that the matrix size ($n \times r$) of **A** could vary from 200×4 to 200×16. These are rank deficient, sparse, ill-conditioned non-square matrixes with no inverse, so a generalized inverse has to be computed to solve the LSE problem. A naïve approach to solving the LSE problem for obtaining the best solution θ^* for θ is to multiply both sides in (8) by $\mathbf{A}^T$ in order to obtain the normal equations representation of the system by computing $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$. This can be done when using high precision floating point units by, for instance, applying LU factorization to $\mathbf{A}^T\mathbf{A}$, but this approach can be numerically very unstable when using reduced precision computation [26]. In fact, we obtained many non-convergence events during performed single precision numerical simulations. In consequence, the heavier but numerically much more robust singular value decomposition (SVD) algorithm was programmed to obtain the Moore-Penrose pseudoinverse of **A** to solve (8)

Experimental processing times of both the SD algorithm execution and the LSE/SVD algorithm execution are summed up in Table 3 for input vector space dimensionalities from $m' = 1$ to $m' = 7$ ($r = 2$ to $r = 128$). These figures correspond to average processing times until convergence on a 100 Mhz clock driven *MicroBlaze* processor. As expected, the majority of the computational load is due to the SVD algorithm. The impact of dimensionality reduction on the learning and adaptation times of the IFANF agent can be clearly observed in this table. For input spaces with 1, 2, and 3 variables, obtained learning times are acceptable for the majority of possible applications of the IFANF agent. For the particular application studied in this paper, processing times for a 4 variable space dimensionality are also acceptable. However, for dimensionalities over 4 variables learning times would be too long for any RT operation and some means of hardware acceleration should be investigated.

Table 3
Processor times for the learning algorithm.

Dimensionality	*MicroBlaze* computing time per learning epoch		
	LSE (by SVD)	SD	Total (LSE + SD)
1 var	2 ms	1.0 ms	3 ms
2 var	62.4 ms	4.6 ms	67 ms
3 var	426.4 ms	18.6 ms	445 ms
4 var	3240 ms	71.8 ms	3,312 ms
5 var	23327 ms	266.1 ms	23593 ms
6 var	170142 ms	954.1 ms	171096 ms
7 var	23.9 min	3334.5 ms	24 min

4.1.2 The PCA Preprocessor

The PCA preprocessor has been implemented by programming the eigenvalue decomposition (EVD) of the covariance matrix of the feature vectors. The EVD is obtained by the Cyclic Jacobi method [25], which requires about $O(d^3+d^2n)$ computations, where d is the dimensionality of the feature space and n is the number of feature vectors or samples used [26]. After deploying and debugging the program in the *MicroBlaze* processor, no numerical issues related to the single precision FPUs have been detected. Experimental timing figures for a 100 Mhz clock driven processor are summed up in Table 4 (Jacobi's algorithm convergence took three iterations in every tested case).

Table 4
Processor times for the principal component analysis.

Input variables	*MicroBlaze* computing time for a complete PCA

	Iterations	Time
5 var	3	5 ms
6 var	3	9 ms
7 var	3	11 ms

4.2 Hardware Partition

The hardware partition implements the PWM-FIS algorithm (7)-(10). Each PWM-FIS core computes one of the agent outputs. The cores communicate with the *MicroBlaze* processor by means of FSL buses –one pair of FSLs per core–. In RT-mode, the microprocessor reads the system inputs, performs vector projection on the reduced base, and sends the transformed inputs to the hardware partition where the agent computes the outputs of the system –both continuous outputs and binary outputs (classifiers) are supported–. The cores compute the incoming signals in parallel to take full advantage of hardware concurrency capability.

The internal architecture of the PWM-FIS cores consists of four main modules, a special-purpose RAM memory, an Address Generation Unit (AGU), the data path, and the core controller. The RAM memory stores the antecedent parameters and the consequent parameters of the cores. It is updated at every parameter adaptation event as a consequence of the learning process. The memory size depends on the number of core inputs, and on the number of antecedents as explained in Section 3.1. The AGU is a simple arithmetic unit that computes the addresses of the active consequents –corresponding to the active rules– in local RAM. The antecedent parameters are accessed by means of a simple multiplexer (MUX); additional arithmetic is not required. It is worthy noting that the restrictions imposed on the antecedent MFs (see Section 3.1) reduce the number of parameters required to define these MFs to only 2(p-1) per system input (i.e. triangle offset and slope in each region). The data path implements the last three layers of the neural network (see Fig.1). It exactly replicates the neuron layout and connections depicted in that figure. Finally, the Core Controller is a very straightforward state machine that controls the core behavior. A detailed description of the internal architecture of the PWM-FIS cores can be found in [24].

Several design specifications have to be taken into account to achieve a balanced compromise between system performance and versatility. On the one hand, processing speed, power consumption, and chip size are critical parameters in the development of a SoC. On the other hand, intelligent agents require sufficiently accurate computation of the processing algorithms involved to achieve good approximation capability and efficient and stable learning/adaptation performances. In the following section, these specifications and related parameters are analyzed in depth.

4.3 Resource occupation and performance

To analyze the resource requirements of the agent implementation a small size *Virtex-5* FPGA, the XC5VSX50T-3 , has been selected. This device has 8.160 slices, 288 digital signal processing blocks (DSPs), 132 RAM memory blocks, and 6 phase locked loops (PLLs). Approximately 10-15 % of resources have to be reserved to implement the *MicroBlaze* processor system. Slices and DSP blocks are key resources in the implementation of the PWM-FIS cores that have been extensively used to implement the intelligent agent efficiently. The resource demand of PWM-ANFIS cores with different sizes has been analyzed. Taking into consideration the results of the previous modeling study described in Section 3, core sizes in the range comprised between one and five inputs have been studied. Fig. 13 depicts the percentage of resources required to implement one PWM-FIS core. As can be seen, the proposed implementation takes full advantage of available DSP blocks as the size of the core increases. A single-input core uses only one DSP, while a five-input core uses almost all available blocks (221 out of 288 DSP blocks); it is thus clear that,

as explained in Section 3.4, dimensionality reduction by PCA preprocessing is an invaluable technique for keeping the implementation figures of the IFANF agent in range.

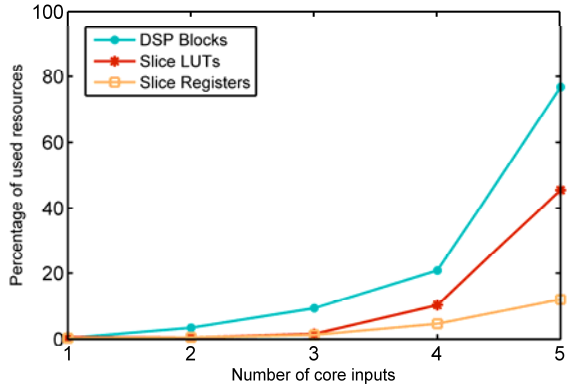

Fig. 13: FPGA resource occupation of each PWM-FIS core as a function of the number of inputs. There is an exponential growth of occupied area with the number of inputs.

Regarding the processing performance, Fig. 14 depicts the maximum allowed frequency for the different core sizes under test. The depicted results have been obtained after place and route. A single-input core could operate at nearly 200 MHz, while a five-input core would barely exceed 50 MHz. As can be seen, each additional input penalizes the core frequency with approximately a 50 MHz reduction. It is worthy to note that these frequencies have been estimated without resource limitations or routing restrictions in the selected FPGA, therefore, an additional performance reduction can be expected after the whole system is implemented in the device. These processing speed figures can be, of course, improved if faster/bigger target devices are selected for implementation.

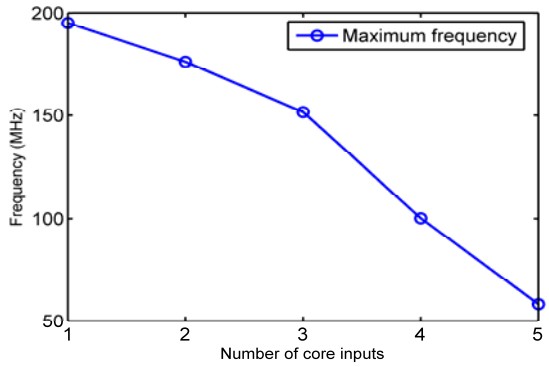

Fig. 14: Maximum achievable clock-frequency for the operation of the FIS cores as a function of the number of inputs.

4.3.1 Power Consumption

Power consumption is critical in most SoC-based applications and it is of particular relevance for an embedded intelligent agent as the reduction of power consumption contributes to its autonomy and reliability. Total power consumption is composed of a static component and a dynamic component. The static power is driven by transistor leakage current, while the dynamic power is based on the switching frequency of used FPGA resources. Although both issues are important, the main problem of present SRAM-based FPGA technology is the static power consumption because, as transistors become increasingly smaller, leakage currents increase. In this sense, a reduction of the logic in the design has a greater impact on power consumption than a reduction of system clock frequency.

With the aim of obtaining a complete picture of the problem, a detailed analysis of power dissipation for different core sizes has been performed. Fig. 15 presents the power

consumption estimation results obtained for fully placed-and-routed designs. As can be seen, the static component is the most important one, while the dynamic component accounts for less than 10% of total power consumption. The increment of power with the number of inputs of the agent presents a similar behavior –an exponential increment– to the resource requirement depicted in Fig. 13.

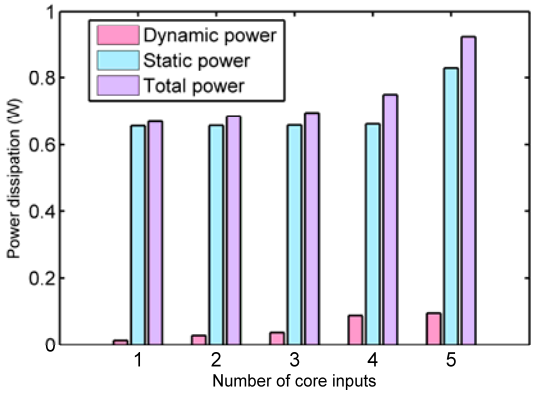

Fig. 15: Estimated power consumption of the complete hardware partition of the IFANF agent (8 FIS cores) as a function of the number of inputs.

4.4 Final implementation

The above design considerations and the experimental results provided in Section 2 and Section 3.3, provide useful information to configure and optimize the architecture of the IFANF agent for the iDorm inhabited environment. Bearing in mind that the device should be selected as small as possible to avoid an excess of power dissipation and make its integration into the environment easier, a good solution for this application example is achieved using eight three-input cores for the HW partition.

The proposed solution was implemented using the XC5VSX50T device. After place and route, this design uses 10452 slice flip-flops (32%), 11367 slice LUTs (34%), 222 DSPs (77%), 99 RAM blocks (75%), and a single PLL. Each one of the eight PWM-FIS cores consumes 1% of the slice flip-flops, 1% of the Slice LUTs, and 9% of DSPs. The remaining resources are required to implement the *MicroBlaze* soft processor and its on-chip subsystem. The 99 RAM memory blocks provide 256 KB of internal memory, 64 KB of data cache, and additional resources to implement the SDRAM memory controller. The whole system comprising both the *MicroBlaze* and the cores, operates with a 100 MHz clock frequency and power estimation consumption gives a total power consumption of 2.08 W; where power dissipated in I/O pads is not included.

5 Experimental results

Finally some graphs obtained from the real operation of the implemented SoPC IFANF agent are presented. These figures show a comparison of the real system performance for some selected meaningful experiments to the equivalent tests performed with Matlab on a PC as described in Section 3.3 (all-software machine precision simulations). This time, however, the random partition of Set1.1 for the initial off-line training has been removed and no validation set is used to improve system's generalization capability. In this manner, experimental tests are performed with exactly the same data in every round thus allowing a fairer comparison between the simulated system and the real system operation. According to Fig. 16, the embedded IFANF agent implemented in the FPGA shows good performance and numerical stability in every tested scenario.

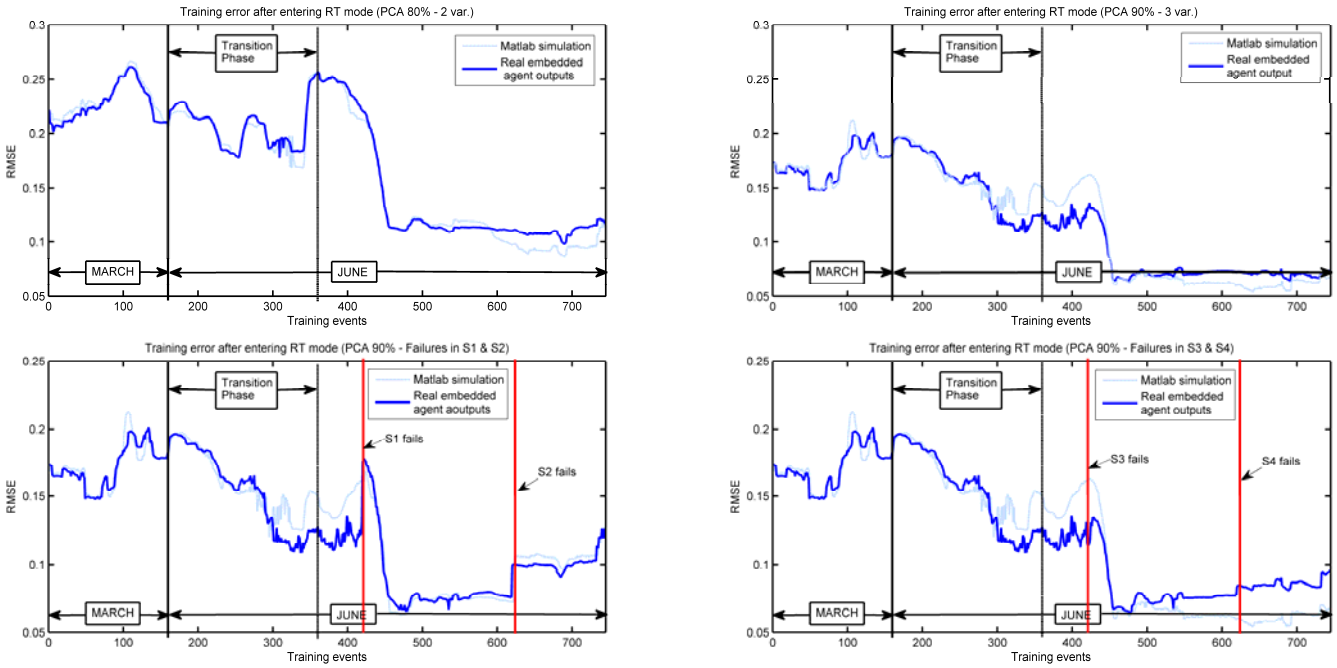

Fig. 16: Verification of the real SoPC IFANF agent operation. Light blue line depicts the modeling error obtained by simulation of the system RT-mode operation on a PC, while dark blue line depicts the modeling error produced by the real system.

The agent's processing times after receiving an input vector are as follows:

1. Projection of the input vector in the new base obtained by PCA plus vector scaling (normalization): 32 μs. This is done by software in the *MicroBlaze.*
2. Data transmission through the FSL bus to the PWM-FIS hardware cores: 0.02 μs (two clock cycles).
3. Neuro-fuzzy information processing in the PWM-FIS hardware cores (eight cores in parallel): 0.08 μs (eight clock cycles).

So the total I/O signal delay of the IFANF agent is 0.1 μs. This, of course, is a much higher processing speed than required for this particular ambient intelligence application, but it clearly demonstrates the potential of this architecture to be applied to the control of complex MIMO systems with very fast response requirements.

6 Conclusions

Neural/fuzzy processing schemes are first-rate computational paradigms to implement intelligence based on data and experience. However, these systems suffer from the so-called curse of dimensionality, which in practice limits their applicability to complex real systems that have to deal with many information channels in parallel. In the case of applications that require operational autonomy with low power consumption and small footprint devices this problem can be even more limiting, since the embedded implementation approach may become impossible to perform. In this work we studied the design issues that come up when facing a SoC embedded neuro-fuzzy intelligent agent implementation for real time autonomous operation by means of an ambient intelligence application based on real world data.

In the first place, we verified that PCA is a very appropriate computational tool for dramatically reducing input space dimensionality in systems that process row data acquired from multisensory deployments in intelligent environments. Combined with a PCA preprocessor, a neuro-fuzzy information processing scheme is able to correctly model

the complex behavior of a human being in a controlled environment using reduced dimensionality input spaces. Moreover, dimensionality reduction is a key process to avoid overfitting and improve the generalization capability of the intelligent agent (prediction capability in the selected application) and to reduce learning and adaptation times.

Secondly, we explored the possibility of using the compressed feature representation provided by PCA to make intelligent agents more robust in the presence of failures in the sensors or in the data communication channels. We showed that PCA is a powerful tool that can rearrange the input space to make the most of the remaining information when successive failures occur, thus providing the agent with robustness and fault tolerance. When applied while system is operating on-line, PCA updates the relevant features that are contained in the information acquired from the sensor network, thus providing the neuro-fuzzy agent with a deep adaptability that is beyond simple parameter adjustment (environment awareness).

Thirdly, from the hardware implementation point of view, dimensionality reduction turned out to be essential to obtain implementable intelligent agents that take advantage of processing parallelism and hardware acceleration. Based on a combination of a neuro-fuzzy processing scheme specially tailored to be implemented on reconfigurable hardware (PWM-FIS cores) and a PCA preprocessor that extracts the meaningful features from available data, we designed a complex intelligent MIMO agent with seven inputs and eight outputs. The agent is able to produce immediate and adequate responses and to self-adapt its parameters in a fraction of a second. This processing speed allows the application of on-line "brute force" parametric learning when modeling performance degradation is detected. Dimensionality reduction has also been the key to achieve a restrained power consumption single chip embedded agent, which widens its usability in many other applications with high-speed reaction and autonomous mobility requirements. Finally, we actually implemented the neuro-fuzzy embedded intelligent agent using the SoC paradigm on an FPGA, tested its performance, and proved the suitability of the proposed architecture for implementing intelligent agents that can be deployed in environments that require ubiquitous and autonomous systems running.

7 Acknowledgments

The authors would like to thank the Basque Government and the Spanish Ministry of Science and Innovation and European FEDER funds for their economic support of this research under Grants IT419-10, IT733-13, S-PC11UN012, S-PC12UN016, and TEC2010-15388. The authors would also like to thank the researchers of the *Intelligent Inhabited Environment Group* for providing the data sets used in this work.

8 References

[1] L. Steels, When are Robots Intelligent Agents?, Robotics and Autonomous Systems, 16 (1995), (1-2) 3–9.

[2] M. Wooldridge and N. R. Jennings, Intelligent agents: theory and practice, The Knowledge Engineering Review ,10 (1995) (2) 115-152.

[3] Y. Meng, An Agent-based Mobile Robot System Using Configurable SoC Technique, in: Proceedings of the IEEE International Conference on Robotics and Automation 2006, ICRA, 3368-3373.

[4] Y. Meng, A Mobile Vision System with Reconfigurable Intelligent Agents, in Proceedings of the International Joint Conference on Neural Networks, 2006, IJCNN '06., 1483-1488.

[5] L. Józwiak, N. Nedjah, and M. Figueroa, Moder development methods and tools for embedded reconfigurable systems: A survey, Integration the VLSI Journal, 43 (2010) (1), 1-33.

[6] Strategic Considerations for Emerging SoC FPGAs, Altera Corp., San Jose, CA, 2011.

[7] I. del Campo, J. Echanobe, G. Bosque, and J. M. Tarela, Efficient Hardware/Software Implementation of an Adaptive Neuro-Fuzzy System, IEEE Transactions on Fuzzy Systems 16 (2008) (3) 761-778.
[8] J.-S. R Jang, ANFIS: Adaptive-Network-based Fuzzy Inference Systems, IEEE Trans. on Systems, Man, and Cybernetics, 23 (1993) 665-685.
[9] N. Kasabov, Evolving Fuzzy Neural Networks for Supervised/Unsupervised Online Knowledge-Based Learning, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 31 (2001) (6) 902-918.
[10] R. Lee, J, Liu, iJADE WeatherMAN; a weather forecasting system using intelligent multiagent-based fuzzy neural network, IEEE Transactions on System, Man, and Cybernetics-Part C, 34 (2004) 369-377.
[11] F. Doctor, H. Hagras, and V. Callahan, A Fuzzy Embedded Agent-Based Approach for Realizing Ambient Intelligence in Intelligent Inhabited Environments, IEEE Transactions on System, Man, and Cybernetics-Part A, 35 (2005) 55-65.
[12] F. Doctor, H. Hagras, and V. Callahan, A Type-2 Fuzzy Embedded Agent to Realise Ambient Intelligence in Ubiquitous Computing Environments, Information Sciences, 171 (2005) 309-334.
[13] K. Figueiredo, M. Vellasco, and M. A. Pacheco, Hierarchical neuro-fuzzy models on reinforcement learning for intelligent agents, in Proceedings of the 8th International Conference on Artificial Neyral Networks: computational Intelligence and Bioinspired Systems, Springer-Berlag Berlin, Berlin, 2005, 424-432.
[14] H. Hagras, F. Doctor, V. Callahan, and A. Lopez, An Incremental Adaptive Life Long Learning Approach for Type-2 Fuzzy Embedded Agents in Ambient Intelligent Environments, IEEE Transactions on Fuzzy Systems, 15 (2007) 41-55.
[15] K. I-K Wang, W. H. Abdulla, and Z. Salcic, Multi-agent System with Hybrid Intelligence Using Neural Network and Fuzzy Inference Techniques, in: H. G. Okuno and M. Ali (Eds.) IEA/AIE, Springer-Verlag Berlin Heidelberg, Berlin, 2007, 473-482.
[16] A. Elfaham, H. Hagras, A Speech Recognizer Based Intelligent Agent For Ambient Intelligent Environments, in Proc. Int. Conf. on Intelligent Environments 2009, IOS Press, 2009, pp. 247–256.
[17] M. H. Fazel Zarandi, E. Hadavandi, and I. B. Turksen, A Hybrid Fuzzy Intelligent Agent-Based System for Stock Price Prediction, Int. J. of Intelligent Systems, 27 (2012) (11) 947-969.
[18] A. K. Yadav and A. K. Sachan, Research and Application of Dynamic Neural Network Based on Reinforcement Learning, in: S. C. Satapathy, P. S. Avadhani, and A. Abraham (Eds.) Proc. Int. Conf. on Information Systems Design and Intelligent Applications, , Springer-Verlag Berlin, Berlin, 2012, 931-942.
[19] A. Traista and M. Elshaw, A Hybrid Neural Emotion Recogniser for Human-Robotic Agent Interaction, in: C. Jayne, S. Yue, and L. Illiadis (Eds.) Engineering Applications of Neural Networks, Springer-Verlag Berlin, Berlin, 2012, 353-362.
[20] S. Sharma, S. Otunba, K. Ogunlana, and T. Tripathy , Intelligent Agents in a Goal Finding Application for Homeland Security, in: K. B. Sundaram and; D. Wu, (Eds.) Proc. IEEE Southeastcon, IEEE, New York, 2012,.
[21] I.T. Jolliffe, Principal Component Analysis, second ed. Springer-Verlag New York, 2002.
[22] J. Shlens, A Tutorial on Principal Component Analysis, Center for Neural Science, NY University, 2009. Available: http://www.snl.salk.edu/~shlens/pub/notes/pca.pdf.
[23] LogiCORE IP MicroBlaze. Micro Controller Sustem (v1.1). Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ds865_microblaze_mcs.pdf.
[24] I. del Campo, K. Basterretxea, J. Echanobe, G. Bosque, and F. Doctor, A System-on-Chip Development of a Neuro-Fuzzy Embedded Agent for Ambient Intelligence Environments, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 42 (2012) (2) 501-512.

[25] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, The Art of Scientific Computing, third edition, Cambridge University Press, 2007.
[26] G. H. Golub and C. F. Van Loan, Matrix Computations, third ed., John Hopkins University Press, Baltimore, 1996.
[27] Virtex 5 Family Overview, Xilinx Inc., San Jose, CA, 2009. Available: http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf

Koldo Basterretxea received the Licenciado degree in Physics with specialization (MSc) in Electronics and Control in 1994 and the Ph.D.degree in Physics in 2002, both from the University of the Basque Country (UPV/EHU), Bilbao, Basque Country, Spain.
He was a Lecturer at the Electronics and Telecommunications Department at the Escuela Universitaria de Ingeniería Técnica Industrial (EUITI) of Eibar (UPV/EHU), from 1995 to 1998, and at the EUITI of Bilbao (UPV/EHU) since 1998, where he currently is a Senior Lecturer. His research interests include digital design of adaptive systems on FPGAs, hardware design for high-speed real-time controllers, neural/fuzzy hardware, hardware/software co-design, and applied soft computing.

Maria Victoria Martínez received the Licenciado Physics Degree with specialization in Electronics and Automatics in 1987 and the Ph.D. Degree in Physics in 2002, both from the University of the Basque Country (UPV/EHU), Bilbao, Spain.
She was a predoctoral research fellow granted by the Basque Government from 1987 to 1988. Currently she is a senior lecturer in the Electricity and Electronics Department of the Faculty of Science and Technology of the UPV/EHU. Her research interest concerns the synthesis and electronic implementation of high-dimensional complex systems with emphasis on efficient realizations in the areas of electronics, device modeling, pattern recognition, computational intelligence and ambient intelligence, among others.

Inés del Campo was born in Buenos Aires, Argentina, in 1961. She received the Licenciado degree in physics with specialization in electronics and automatics in 1987 and the Ph.D. degree in physics, in 1993, both from the University of the Basque Country (UPV/EHU), Bilbao, Spain.
Currently she is a Senior Lecturer in the Electricity and Electronics Department of the Faculty of Sciences and Technology of the UPV/EHU. She has published articles in international journals and conferences in the areas of electronics, computational intelligence, intelligent control, ambient intelligence, and pattern recognition, among others. Her research interests mainly concern system-on-chip (SOC) design, hardware/software codesign, reconfigurable hardware, pervasive computing, artificial neural networks (ANNs), fuzzy systems, and genetic algorithms. She is also interested in the internet of things and its application in the context of ubiquitous computing and ambient intelligence.

Javier Echanobe received the Licenciado degree in physics from the University of the Basque Country (UPV/EHU), Spain, and the Ph.D. degree from the University of Navarra, Pamplona, Spain, in 1990 and 1998, respectively.
He was a Predoctoral Researcher (granted by the Basque Government) from 1992 to 1996. He has been an Associate Professor in the Department of Electricity and Electronics, UPV/EHU, from 1999 to 2009. Since 2009 he is a Senior Lecturer in that department. His research interests focus on 1) digital electronics: embedded systems, reconfigurable FPGAs, DSPs, SoPC; 2) computational intelligence: artificial neural networks, fuzzy systems, and neuro-fuzzy systems; 3) ubiquitous computing: ambient intelligence, intelligent environments. Dr. Echanobe has published many papers in international journals and conferences in most of those areas.