Evolutionary Computing and Particle Filtering: A Hardware-Based Motion Estimation System

Alfonso Rodríguez and Félix Moreno, *Member, IEEE*

Abstract—Particle filters constitute themselves a highly powerful estimation tool, especially when dealing with non-linear non-Gaussian systems. However, traditional approaches present several limitations, which reduce significantly their performance. Evolutionary algorithms, and more specifically their optimization capabilities, may be used in order to overcome particle-filtering weaknesses. In this paper, a novel FPGA-based particle filter that takes advantage of evolutionary computation in order to estimate motion patterns is presented. The evolutionary algorithm, which has been included inside the resampling stage, mitigates the known sample impoverishment phenomenon, very common in particle-filtering systems. In addition, a hybrid mutation technique using two different mutation operators, each of them with a specific purpose, is proposed in order to enhance estimation results and make a more robust system. Moreover, implementing the proposed Evolutionary Particle Filter as a hardware accelerator has led to faster processing times than different software implementations of the same algorithm.

Index Terms—Embedded systems, evolutionary computing, FPGAs, particle filtering.

1 Introduction

STATE estimation and prediction are considered major concerns in the field of real-world system analysis. A large number of studies regarding these two hot topics can be found throughout the literature, ranging from the basic filtering approaches to more developed alternatives that make use of complex mathematical concepts.

In 1960, a new filtering algorithm was proposed: the Kalman Filter [1]. This algorithm estimates unknown states using noisy measurements in two stages: prediction and correction. In the former, the system predicts new states, which are then corrected in the latter using the measurements. The results obtained with this double-staged approach are more precise than those obtained from pure observation. The algorithm has been thoroughly analyzed since it was proposed, and additional resources can be found in the literature in this regard [2].

Limitations in the Kalman Filter algorithm when dealing with systems other than Gaussian and linear revealed that a different alternative had to be developed in order to cope with more complex estimations. Monte-Carlo simulations, as well as new mathematical models, such as Hidden Markov Models (also used in Kalman filtering), provided a new framework in which new algorithms, the so-called particle filters, were developed. An extensive report on these topics can be found in [3]. The most representative implementation of a particle filter appears in [4], where a new algorithm, the Bootstrap Filter, is presented. This filter is usually thought of as the reference particle filter.

This work addresses two different enhancements over the basic particle filter algorithm: on the one hand, a fast hardware-based implementation intended for embedded computations; on the other hand, error mitigation through the addition of evolutionary capabilities within the particle-filtering algorithm. Hardware-based architectures can boost particle filters performance, since parallel processing capabilities are able to speed up many repetitive tasks. Some approaches have been developed in the literature. For example, in [5] a hardware implementation of the Bootstrap Filter is presented. The authors use VHDL (Very high speed integrated circuit Hardware Description Language) and a Xilinx Virtex-2 FPGA (Field Programmable Gate Array) in order to obtain real-time properties in an object tracking application. In [6], the same particle-filtering algorithm is implemented in a Xilinx Virtex-5 FPGA. Parallel processing capabilities are used in order to process data concurrently, since every stage performs its computation while the others are working with other valid data, i.e. in pipeline-fashion. This leads to a significant decrease in terms of resource consumption. However, there are not only hardware approaches, but also software/hardware mixed implementations. For instance, in [7] a SoPC (System on Programmable Chip) model with hardware accelerators is proposed, targeting an Altera Cyclone II FPGA. An embedded processor is in charge of computing weight values, whereas the particle update is carried out in hardware, since it is a highly repetitive process. The authors also introduce some characteristic elements from evolutionary computation, such as tournament selection algorithms, in the resampling stage. Another FPGA-based implementation of a particle filter can be found in [8]. In this particular example, the algorithm is enhanced using color histogram optimizations. A Xilinx Virtex-5

FPGA performs all weight and histogram calculations taking advantage of parallel processing capabilities.

However, in the last few years there has been an increase in the number of works related with high-performance particle filters, which use huge numbers of particles in order to solve complex high-dimensional problems. These specific problems require massive amounts of computational resources. Hence, some approaches have started to use many-core architectures in order to cope with that restriction. Parallel computing has proved itself a feasible alternative to traditional approaches, as it has been shown in [9], where the authors establish a comparison between a classic CPU (Central Processing Unit) and a GPU (Graphics Processing Unit). Their results show that the more parallel the approach is, the faster the processing can be done. However, there are some limitations, since each parallel block resamples from a subpopulation and not from the whole population. Different parallel implementations using GPGPUs (General Purpose GPUs) and multi-core CPUs also appear in [10]. The authors provide a detailed report on sensitivity analysis when scaling system parameters such as the number of particles per filter, the number of subfilters and even the state dimensions. The authors have also extended their research to real-time control applications of distributed computing approaches to particle filtering, as in [11].

Particle filtering and evolutionary algorithms, especially genetic algorithms, share many conceptual similarities. These similarities have been studied and documented for a long time. For instance, in [12] a modified particle filter is presented. The author uses genetic operators such as crossover and mutation to implement the prediction stage of the filter. The connection between particle filtering (Monte-Carlo simulations) and Bayesian inference and their application to evolutionary environments has also been studied. In [13], the foundations for Bayesian evolutionary computation are presented. The idea is to let Bayes rule guide the evolutionary process, since it is reasonable to think that the most probable solution is the best one. However, this is not state-of-the-art research.

In the last few years, the main interest of evolutionary computation applied to particle filtering has been focused on mitigating sample impoverishment phenomena, which is due to suboptimal resampling strategies. Evolutionary algorithms, as stated in previous sections, use genetic operators in order to transfer good genes and to introduce genetic diversity. This last feature is the key to understand why evolutionary computing is suitable for solving sample impoverishment (i.e. diversity loss) problems. Hence, in [14] an evolutionary approach is introduced in order to mitigate those effects. The authors propose introducing genetic operators right after performing the importance sampling stage and immediately before the resampling stage. The resampling stage, as opposed to the basic Bootstrap Filter [4], performs an adaptive strategy, in which a parameter (effective number of particles) is considered in order to decide whether to perform resampling or not. In this paper, it is also shown that the mutation operator provides better dynamic response when the state jumps abruptly. Other research lines use parallel distributed filters, i.e. with several particle subpopulations evolving at the same time. In [15], the authors use these subpopulations to perform genetic operations and, from time to time, migrate the best individuals from one subpopulation to the others so that the best genetic traits can be shared. This also leads to an improvement in global optimum search. Moreover, the concept of genetic resampling stage is introduced in this paper for the first time. Nevertheless, the paper provides only simulation results.

More complex examples can also be found in the literature: a hybrid evolutionary particle filter is presented in [16]. This hybrid approach tries to take advantage of both genetic algorithms (to maintain particle diversity) and particle swarm optimization (to optimize the final particle distribution). Furthermore, the algorithm presented has parallel features, thus reducing computation times. The strategy presented divides the population in two groups, and then performs the specific operations that are required: in one group, a genetic algorithm; in the other, particle swarm optimization. Before the next time step, a migration operation is performed (to share genetic information, as in previous examples).

Real-world applications include object tracking, as in [17], where another evolutionary approach is presented in order to deal with sample impoverishment. In this case, the evolutionary resampling stage may or may not take part in the estimation loop. The decision is made based on the aforementioned parameter, i.e. the effective number of particles. The algorithm presented provides good results but it is not accurate in tracking sequences with occlusion. Recent studies with differential evolution have been conducted in order to reduce significantly sample sizes [18]. The differential evolution algorithm divides the particles in three different groups: the first group would undergo crossover; the second group would undergo mutation; the last group would not suffer any modification. One important remark about this work is that the evolutionary stage takes place at the very beginning of the process, and it is immediately followed by the importance sampling stage.

In conclusion, evolutionary computation has found a vast field of application in which optimization is not the main concern. The characteristic properties of the genetic operators make evolutionary algorithms a potential tool in particle filtering, since they are able to reduce to almost negligible values both problems: particle degeneracy (it is avoided performing resampling) and sample impoverishment (it is avoided introducing genetic diversity in the particle population).

The rest of this paper is organized as follows: first, the mathematical concepts regarding particle filtering are thoroughly covered in Section 2. In Section 3, the evolutionary algorithm that has been implemented in the resampling stage is discussed. The proposed system architecture is presented in Section 4. Section 5 includes experimental results, followed by the conclusions, presented in Section 6.

2 Particle Filtering

Particle filters are based upon Monte-Carlo methods, which are a set of computational algorithms that obtain numerical results by using repetitive random sampling, and Bayesian inference, which uses the well-known Bayes rule to compute the posterior distribution from the previous distribution and the likelihood distribution. Bayes rule, which is also referred to as Bayes theorem, is often expressed as (1).

$$P(a|b) = \frac{P(b|a)\,P(a)}{P(b)} \tag{1}$$

$P(a|b)$ represents the posterior distribution, i.e. the probability of a when b is known to be a particular value, $P(a)$ is the prior distribution, $P(b|a)$ is the likelihood distribution, i.e. the probability of obtaining b after a has been observed, and $P(b)$ is the marginal likelihood, which is independent of the hypothesis that is being tested (i.e. changes in the hypothesis do not affect the marginal likelihood). Bayesian inference is nothing but an updating process in which not only evidence (prior distribution) but also previous knowledge (likelihood distribution) are taken into account in order to draw a conclusion.

All equations, expressions and conclusions in this section are based upon the contents of [19].

2.1 Perfect Monte-Carlo Sampling

Hereafter, let us assume that real-world systems can be expressed as Markovian, non-linear, non-Gaussian state-space models. A Markov model is a stochastic model (i.e. a system which shows random behavior) in which the Markov property is satisfied, i.e. future states depend only upon the present state and not the previous history of the system. Markov property is also referred to as memoryless property.

$$p(x_t|x_{t-1}) \tag{2}$$
$$p(y_t|x_t) \tag{3}$$

(2) represents the hidden states transition model, or process model, whereas (3) represents the observation or measurement model. These models are called Hidden Markov Models (HMMs) because the state sequence is unknown, i.e. not all states are observable. The hidden states at time t are expressed as x_t, whereas the observations at time t are expressed as y_t. With these definitions, the posterior distribution is calculated using Bayes theorem as in (4).

$$p(x_{0:t}|y_{1:t}) = \frac{p(y_{1:t}|x_{0:t})\,p(x_{0:t})}{\int p(y_{1:t}|x_{0:t})\,p(x_{0:t})\,\mathrm{d}x_{0:t}} \tag{4}$$

$p(x_{0:t}|y_{1:t})$ is the posterior distribution of the states $x_{0:t}$ having observed the measures $y_{1:t}$. The subscripts represent the time steps in which the distribution is defined, e.g. $p(x_{0:t}|y_{1:t})$ is the posterior distribution of the hidden states from $t = 0$ to $t = t$ when the observations are known from $t = 1$ to $t = t$.

2.2 Importance Sampling

Perfect Monte-Carlo sampling is unfeasible in most real-world situations, since it is impossible to sample from the posterior distribution. In those cases, another sampling strategy called Importance Sampling (IS) is used. With this approach, the distribution from which the samples are drawn is not the actual posterior distribution, but an arbitrary user-defined distribution called proposal distribution, importance function, or importance sampling distribution. Therefore, each drawn sample has to be weighted in order to make both posterior and importance sampling distributions match. Hence, IS can be expressed in terms of the equations (5) and (6).

$$\pi(x_{0:t}|y_{1:t}) \tag{5}$$
$$w(x_{0:t}) = \frac{p(x_{0:t}|y_{1:t})}{\pi(x_{0:t}|y_{1:t})} \tag{6}$$

(5) is the importance sampling distribution and (6) are the importance weights. The posterior distribution is then approximated by a finite set of particles using the expression in (7).

$$p(x_{0:t}|y_{1:t}) \approx \hat{P}_N(\mathrm{d}x_{0:t}|y_{1:t}) = \sum_{i=1}^{N} \tilde{w}_t^i \, \delta_{x_{0:t}^i}(\mathrm{d}x_{0:t}) \tag{7}$$

$\hat{P}_N(\mathrm{d}x_{0:t}|y_{1:t})$ is the approximated posterior distribution, $\delta_{x_{0:t}^i}(\mathrm{d}x_{0:t})$ is the delta-Dirac function located in $\mathrm{d}x_{0:t}$, N is the number of particles and $\tilde{w}_t^i$ is the normalized weight for particle i up to time t, which is obtained from the particle weight $w(x_{0:t}^i)$ using (8).

$$\tilde{w}_t^i = \frac{w(x_{0:t}^i)}{\sum_{j=1}^{N} w(x_{0:t}^j)} \tag{8}$$

2.3 Sequential Importance Sampling

One of the most important disadvantages of the aforementioned IS strategy is that it is computationally intensive, since the posterior distribution is calculated from scratch each time step. Sequential Importance Sampling (SIS) performs the same computations but in a recursive manner. The importance sampling distribution and the weight update equation are now expressed as (9) and (10).

$$\pi(x_{0:t}|y_{1:t}) = \pi(x_{0:t-1}|y_{1:t-1})\,\pi(x_t|x_{0:t-1}, y_{1:t}) \tag{9}$$
$$w(x_t) \propto w(x_{t-1})\,\frac{p(y_t|x_t)\,p(x_t|x_{t-1})}{\pi(x_t|x_{0:t-1}, y_{1:t})} \tag{10}$$

Notice that these expressions provide a simple methodology for online filtering, since particle weights at any

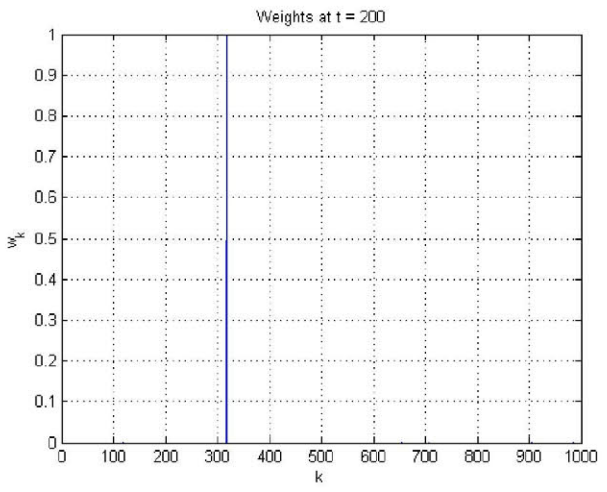

Fig. 1: Particle degeneracy phenomenon. Notice that after a finite estimation time, the particle population is biased, since only one individual has a non-zero weight. There is no diversity in the population, and thus posterior distributions are not accurate.

current time step are obtained from the value at the previous time step. However, these equations are still very complex, thus leading to performance problems if real-time requirements are present as design constraints. A simple approach can be adopted in order to overcome these limitations. By selecting the prior distribution as the importance function, the equations are reduced significantly to (11) and (12).

$$\pi(x_{0:t}|y_{1:t}) = p(x_{0:t}) = p(x_{0:t-1})\,p(x_t|x_{t-1}) \tag{11}$$

$$w(x_t) \propto w(x_{t-1})\,p(y_t|x_t) \tag{12}$$

Therefore, the transition model $p(x_t|x_{t-1})$, i.e. the process model, is used in the sampling stage, and the observation model $p(y_t|x_t)$, i.e. the measurement model, in the weight update stage.

Basic particle-filtering implementations using SIS show biased populations, where only one particle is significant (i.e. its weight is one, whereas the rest of the population is negligible) after a certain finite execution time, as it can be seen in Fig. 1. This is the so-called particle degeneracy phenomenon, and it might lead to large errors in the estimations, since the posterior distribution is represented by only one effective particle. The main reason for this phenomenon to appear is that the variance of the importance weights can only increase over time, as it has been shown in [20].

2.4 Resampling

The particle degeneracy phenomenon can be mitigated provided that an additional resampling stage is included in the basic particle filter. The resampling stage is located after the sampling and weight calculation processes, and generates a new particle population according to their current weights. This new population can then be homogenized by making all particle weights equal. This

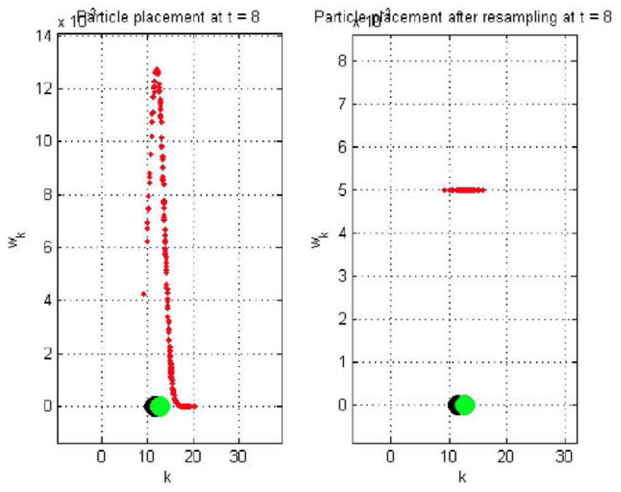

Fig. 2: Sample impoverishment phenomenon. Suboptimal resampling strategies generate inaccurate posterior distributions. Results show large variations in weight and particle distributions before (left) and after (right) resampling operations. In the images, actual measurements appear in black, estimated states in green and particle distributions in red.

is the normal procedure in the Bootstrap Filter [4], whose algorithm is shown in Table 1.

However, suboptimal resampling algorithms tend to generate populations where no diversity exists, for particles with higher weights are statistically selected many times [21], which means that most of the particles are identical (see Fig. 2). Therefore, the posterior distribution is inaccurate, and estimation performance is significantly reduced. This phenomenon is known as sample impoverishment, since the population is reduced to a set of particles (usually a small number) with the same state, not providing additional information.

Both particle degeneracy and sample impoverishment are the most important problems in particle-filtering systems, and have to be mitigated, if not eliminated.

TABLE 1
Bootstrap Filter Algorithm

```
 0:  Initialize particle population $x_o^i$ and $k = 1$
 1:  while(1)
 2:      for ($i = 0$; $i < PARTICLES$; $i$++)
 3:          Importance sampling $x_k^i \sim p(x_k^i|x_{k-1}^i)$
 4:          Compute weights $w(x_k^i) \sim p(y_k^i|x_k^i)$
 5:          Normalize weights $\tilde{w}_k^i = \frac{w(x_k^i)}{\sum_{j=1}^{PARTICLES} w(x_k^j)}$
 6:      end for
 7:      for ($i = 0$; $i < PARTICLES$; $i$++)
 8:          Resample according to weight $x_k^i \propto \tilde{w}_k^i$
 9:      end for
10:      $k$++
11:  end while
```

The Boostrap Filter algorithm presents some similarities with the Kalman Filter. There is a prediction stage, in which each particle is updated using the prior distribution (importance sampling). The update is carried out afterwards, first computing the normalized weights and then performing resampling in order to reject those individuals whose weight is smaller. Notice that there is a resampling operation each time step k.

2.5 Further Discussion

Even though HMMs are very widely used throughout the literature, there are other mathematical models that provide much more generalization and in which particle filtering is still feasible. That is the case of Pairwise and Triplet Markov Models.

Pairwise Markov Models (PMMs) can be built upon the assumption that the pair (x_t, y_t) is markovian, whereas Triplet Markov Models (TMMs) consider the triplet (x_t, r_t, y_t), where r_t is an additional auxiliary process, to be markovian. In [22], the authors prove that TMMs are more general than PMMs and PMMs more general than HMMs. Furthermore, the authors show that classical particle filtering approaches can be extended to both PMMs and TMMs.

Examples of particle filters implemented using these more general models as mathematical foundation can be found in the literature [23]. However, and since the objective of this work is to show the benefits of both evolutionary computing and hardware acceleration in particle filtering, the classical HMMs approach is sufficient.

3 Evolutionary Resampling

The sample impoverishment phenomenon in particle filtering appears, as it has been mentioned in the previous section, when the posterior distribution is represented by a particle set in which there is no diversity. Since one of the key features of genetic operators is introducing genetic diversity, it seems reasonable to combine evolutionary computation and particle filtering in order to improve the posterior distribution precision, and thus, the overall filtering performance.

It is for this reason that the resampling stage of the proposed Evolutionary Particle Filter features a genetic algorithm. Each chromosome encodes the particle state, as it is shown in (13).

$$\boldsymbol{x_k^i} = \begin{bmatrix} x_k^i \\ y_k^i \\ v_{x_k}^i \\ v_{y_k}^i \end{bmatrix} \tag{13}$$

x_k^i, y_k^i are the position values in both x-axis and y-axis, and $v_{x_k}^i$, $v_{y_k}^i$ are the velocity values referred to the same axes. The subscripts k represent the current time step, whereas the superscripts i represent the particle identifier.

3.1 Genetic Operators

Let us now focus on the genetic operators: crossover and mutation. Crossover not only increases population diversity, but also helps improving population fitness, for children that inherit the best characteristics, i.e. genes, from the parents are more likely to pass from one generation to the next one. Since all state variables are represented using real numbers, arithmetic crossover can be applied in order to combine genetic traits from both parents. Crossover operations are therefore expressed in terms of the equations in (14).

$$\begin{cases} \boldsymbol{x_k^a} = \alpha\, \boldsymbol{x_k^i} + (1-\alpha)\, \boldsymbol{x_k^j} \\ \boldsymbol{x_k^b} = \alpha\, \boldsymbol{x_k^j} + (1-\alpha)\, \boldsymbol{x_k^i} \end{cases} \tag{14}$$

$\boldsymbol{x_k^a}$, $\boldsymbol{x_k^b}$ are the offspring particles obtained from the parents $\boldsymbol{x_k^i}$ and $\boldsymbol{x_k^j}$, and $\alpha \sim U(0,1)$. The superscripts are the individual identifiers (a, b for the children; i, j for the parents).

Mutation, on the other hand, is used to introduce random variations within the particle population, which also increases population diversity. In this work, two different mutation proposals have been defined: local search mutation and random placement mutation. The former mutates the parent and generates a child in a close region around the state of the parent, whereas the latter generates a valid random particle in the state space, i.e. within its limits. These two possible mutation operations provide more flexibility. Local search mutation is defined as in (15).

$$\boldsymbol{x_k^c} = \boldsymbol{x_k^i} + \boldsymbol{\delta_k} \tag{15}$$

$\boldsymbol{x_k^c}$ is the particle generated after mutation, $\boldsymbol{x_k^i}$ are the state variables of the parent i, and $\boldsymbol{\delta_k}$ is an array where each component is defined by $\delta_{ki} \sim N(\mu_i, \sigma_i)$. Random placement mutation is based upon (16).

$$\boldsymbol{x_k^c} = \boldsymbol{x_{min}} + \beta\,(\boldsymbol{x_{max}} - \boldsymbol{x_{min}}) \tag{16}$$

$\boldsymbol{x_k^c}$ is, again, the particle generated after mutation, $\boldsymbol{x_{min}}$ and $\boldsymbol{x_{max}}$ are the minimum and maximum allowable values for the state variables, i.e. the state space boundaries, and $\beta \sim U(0,1)$.

Why have these genetic operators, and not others, been selected? Arithmetic crossover generates children by averaging, with some weighting process in between, the parents states. Therefore, if the parents are in a close region around the maximum fitness point, the children will likely have better fitness values. Otherwise, they will be discarded in future generations. Hence, the posterior distribution will be more precise. The proposed mutation strategies cover two different and opposed situations. On the one hand, local search mutation is useful when the particle population is close to the optimal value, since mutation generates another particle in a close environment of the parent. This leads to an increase in high-fitness population. On the other hand, random placement mutation provides robustness in those time steps in which the tracking is lost, i.e. the posterior distribution is inaccurate or cannot be properly estimated.

3.2 Fitness Function

Given the fact that particle filtering and evolutionary computing have common features, such as being population-based algorithms, some parallelisms can be established (at least to some extent). For instance, weights can also be thought of as fitness values. By making this simple assumption, the design is simplified significantly, since only one computation (weight) has to be performed instead of two (fitness value and weight). In this particular case, particle weights (i.e. fitness values) are calculated using a normal multivariate probability density function, using two dimensions and assuming that the two random variables are non-correlated. Therefore, these values can be obtained using (17).

$$f(x,y) = \frac{1}{2\,\pi\,\sigma_x\,\sigma_y} \exp -\frac{1}{2}\left(\frac{(x-\mu_x)^2}{\sigma_x^2} + \frac{(y-\mu_y)^2}{\sigma_y^2}\right) \quad (17)$$

σ_x and σ_y are the standard deviation values of the random variables x and y, which are the actual measurements in both x-axis and y-axis, whereas μ_x and μ_y represent the mean values, which are the current particle x-axis and y-axis estimated measurements (calculated from the particle state variables). The fitness function is then expressed as (18).

$$f(\boldsymbol{z_k}, \boldsymbol{\hat{z}_k^i}) = \frac{1}{2\,\pi\,\sigma_x\,\sigma_y} \exp -\frac{1}{2}\left(\frac{(z_{xk} - \hat{z}_{xk}^{\ i})^2}{\sigma_x^2} + \frac{(z_{yk} - \hat{z}_{yk}^{\ i})^2}{\sigma_y^2}\right) \quad (18)$$

Therefore, each particle weight/fitness is the probability of obtaining the measured positions in a normal 2-dimensional probability density function whose mean values are that particle positions.

3.3 Selection

In traditional implementations of evolutionary algorithms, parent selection usually follows stochastic processes, whereas survivor selection is based upon deterministic processes. In this work, however, both parents and survivors are selected using Stochastic Universal Sampling (SUS), since having only one selection algorithm is more suitable for hardware implementations with limited resources. This algorithm performs fitness proportionate selection in a roulette-wheel fashion. However, as opposed to the basic roulette-wheel implementation, it is an unbiased process in which any individual may be selected, even those with the lowest fitness values in the population. SUS algorithm is shown in Fig. 3.

4 System Architecture

A novel hardware-based Evolutionary Particle Filter, the so-called HW-EPF, is presented in this work. The algorithm (see Table 2) has been implemented in a FPGA using VHDL. The HW-EPF is used as a hardware accelerator by the main processor of a SoPC, since it provides better performance than software in fast and repetitive sequential processes, e.g. particle update or particle sorting. The block diagram of the SoPC can be seen in Fig. 4.

TABLE 2
HW-EPF Algorithm

```
0:  Initialize particle population $x_o^i$ and $k = 1$
1:  while(1)
2:    for ($i = 0$; $i < PARTICLES$; $i$++)
3:      Importance sampling $x_k^i \sim p(x_k^i | x_{k-1}^i)$
4:      Compute weights $w(x_k^i) \sim p(y_k^i | x_k^i)$
5:      Normalize weights $\tilde{w}_k^i = \frac{w(x_k^i)}{\sum_{j=1}^{PARTICLES} w(x_k^j)}$
6:      for ($gen = 0$; $gen < GENERATIONS$; $gen$++)
7:        Parent selection (SUS)
8:        Draw $r \sim U(0,1)$. Arithmetic crossover if $r < p_{cross}$
          Draw $r \sim U(0,1)$. Mutation if $r < p_{mut}$
9:          Local search mutation if $r \geq p_{mut}\, r_{mut}$
            Random placement mutation if $r < p_{mut}\, r_{mut}$
10:       Survivor selection (SUS)
11:     end for
12:   end for
13:   $k$++
14: end while
```

The HW-EPF algorithm modifies the basic resampling stage and includes a genetic algorithm (steps 6 to 11) in order to take advantage of the optimization capabilities of evolutionary computation. In addition, more than one mutation operator has been included, in order to further improve estimation performance.

The proposed implementation takes advantage of the distributed resources inside the FPGA. For instance, particle states are stored in the internal RAM (Random Access Memory), and the fitness function is stored in internal LUTs (Look-Up Tables). In addition, some operations have been multiplexed in order to maximize resource sharing, i.e. to minimize resource consumption, which also provides significant benefits (e.g. less area overhead). Hence, a tradeoff between resource utilization and hardware acceleration is made, which is a common practice in embedded system design.

Limited-precision fixed-point arithmetic is inherent to most hardware designs, since computations require fewer resources and are carried out faster. Some problems may arise when using limited-precision data types,

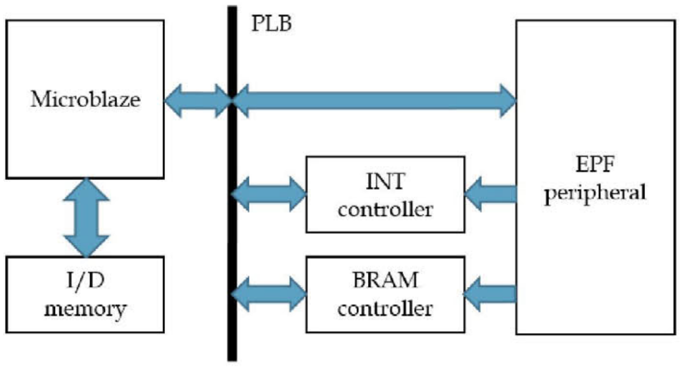

Fig. 4: SoPC internal architecture. The main processor in the system (in this case a Xilinx MicroBlaze soft-core) uses the HW-EPF as a hardware accelerator. The HW-EPF provides three different interfaces: a register-based control port via PLB (Processor Local Bus), a memory controller port and some interrupt signals.

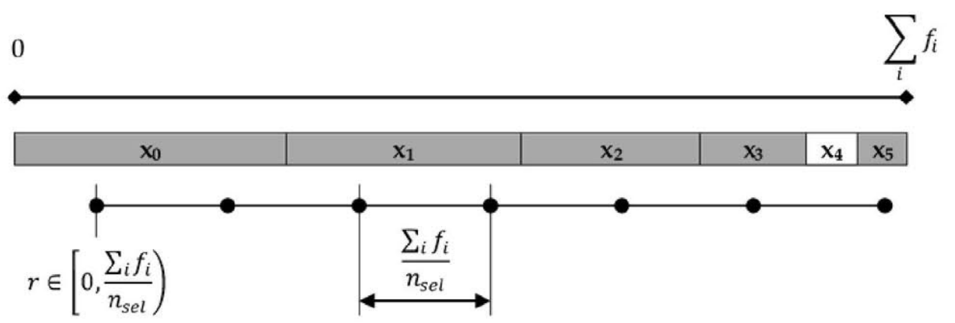
0
$\sum_i f_i$
x0
x1
x2
x3
x4
x5
$\frac{\sum_i f_i}{n_{sel}}$
$r \in \left[0, \frac{\sum_i f_i}{n_{sel}}\right)$

Fig. 3: Stochastic Universal Sampling. After sorting the particles according to their fitness values (f_i) and computing the cumulative fitness function, individuals are selected drawing a uniform random number r and dividing the whole cumulative fitness range in as many equal divisions as individuals to be selected (n_{sel}).

e.g. overflow, underflow, and the results might not be accurate enough. However, these factors have no relevant influence in the proposed architecture, as it will be shown in forthcoming sections.

4.1 Motion Model Equations

Assuming uniform motion patterns in both axis, the importance sampling process, i.e. particle updating process, is performed using the system of linear equations shown in (19) and (20).

$$\boldsymbol{x_k^i} = A\,\boldsymbol{x_{k-1}^i} + \boldsymbol{w_{k-1}} \quad (19)$$

$$A = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (20)$$

$\boldsymbol{w_{k-1}}$ is an array where each component is defined by $w_{i\,k-1} \sim N(\mu_i, \sigma_i)$, i.e. an array of normal-distributed random numbers or white noise. The coefficient T in matrix A represents the sampling time, i.e. the theoretical interval between two consecutive time steps, and has been included in order to weight the influence of the particle velocities in the estimation results. Notice that this parameter is not related with the system clock frequency at all.

In order to reduce the overall number of multipliers, each particle uses four clock cycles (one clock cycle per state variable) to be fully updated. Multiplexing the input signals of the arithmetic operators (adders and multipliers), and thus providing resource sharing mechanisms, resource consumption remains balanced.

With the updated states, the system provides an estimated measurement. This value is computed using the expressions (21) and (22).

$$\boldsymbol{\hat{z}_k^i} = \begin{bmatrix} \hat{z}_{x\,k}^{\,i} \\ \hat{z}_{y\,k}^{\,i} \end{bmatrix} = H\,\boldsymbol{x_k^i} \quad (21)$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad (22)$$

$\hat{z}_{x\,k}^{\,i}$ and $\hat{z}_{y\,k}^{\,i}$ are the estimated measured values of the particle i state variables at time k in both x-axis and y-axis respectively.

Since the estimated measurements are the position state variables, the architecture has been simplified avoiding the usage of additional resources.

4.2 Genetic Operators

Genetic operations are performed over each parent if a uniform random number is below the established thresholds, i.e. crossover and mutation probabilities. Crossover unit follows equation (14), whereas mutation unit follows equations (15) and (16). As far as the hardware implementation of these modules is concerned, they share the same architecture as the one described in the previous subsection. The arithmetic units have their inputs multiplexed in order to favor resource sharing, at the expense of adding a fixed latency, in terms of clock cycles, to the system.

4.3 Fitness Evaluation

Implementing equation (18) is not feasible in hardware, due to the limitations of fixed-point arithmetic. In addition, complex mathematical functions, such as exponentials, are highly time-consuming. Therefore, they should not be used when optimizing system performance, even if a floating-point unit is available in the design. In order to overcome this disadvantage, the proposed implementation changes complex computations by indexing operations in a LUT. The values of equation (18) are evaluated in a small set of reference points at synthesis time. The result of this discretization process is shown in Fig. 5. Then, whenever the system needs to compute a value, the LUT is accessed at run-time and the fitness is obtained by zero-order interpolation. With this proposed alternative, the system shows improved estimation times without affecting estimation precision.

4.4 Estimation

At the end of each estimation step, the current state is obtained from the whole particle population using the expression (23).

$$\hat{\boldsymbol{x}}_{\boldsymbol{k}} = \begin{bmatrix} \hat{x}_k \\ \hat{y}_k \\ \hat{v_x}_k \\ \hat{v_y}_k \end{bmatrix} = \sum_{i=1}^{N} \tilde{w}_k^i \, \boldsymbol{x}_{\boldsymbol{k}}^{\boldsymbol{i}} \tag{23}$$

$\hat{\boldsymbol{x}}_{\boldsymbol{k}}$ is the estimated state at time k, $\boldsymbol{x}_{\boldsymbol{k}}^{\boldsymbol{i}}$ are the state variables of the particle i at time k, and $\tilde{w}_k^i$ are the normalized weights for each particle, computed as in (24).

$$\tilde{w}_k^i = \frac{w(x_k^i)}{\sum_{j=1}^{N} w(x_k^i)} = \frac{f(\boldsymbol{z}_{\boldsymbol{k}}, \hat{\boldsymbol{z}}_{\boldsymbol{k}}^{\boldsymbol{i}})}{\sum_{j=1}^{N} f(\boldsymbol{z}_{\boldsymbol{k}}, \hat{\boldsymbol{z}}_{\boldsymbol{k}}^{\boldsymbol{j}})} \tag{24}$$

4.5 Random Number Generator

Particle filters are always built upon stochastic processes. Furthermore, evolutionary algorithms also require random numbers in order to decide whether to apply a specific genetic operator, or which individuals are chosen in the selection stages. Hence, random number generation is considered a main task within the HW-EPF.

Taking a closer look at the proposed algorithms, it is clear that two different types of random numbers have to be generated: on the one hand, uniform-distributed random numbers, e.g. crossover and mutation probability thresholds, α in (14) and β in (16). On the other hand, normal-distributed random numbers, e.g. $\boldsymbol{w}_{\boldsymbol{k-1}}$ in process model equation (19), or $\boldsymbol{\delta}_{\boldsymbol{k}}$ in local search mutation equation (15).

Uniform distributions are relatively easy to obtain in hardware systems using a LFSR (Linear Feedback Shift Register). These devices generate a stream of uniform-distributed pseudorandom numbers. Since motion estimation is not a highly demanding application, pseudorandom features are enough.

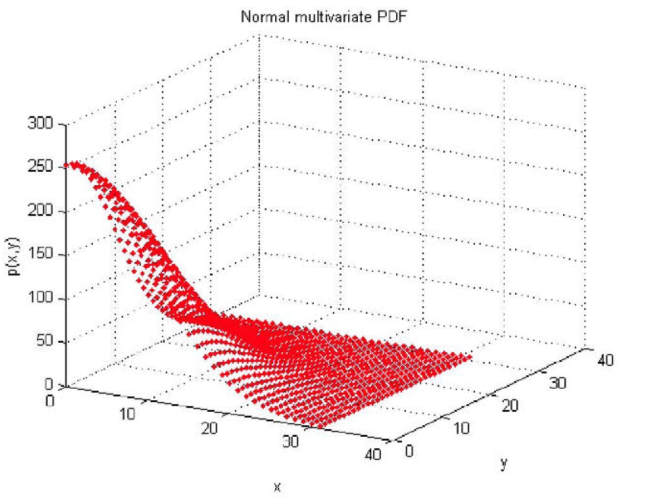

Fig. 5: Fitness evaluation LUT. The normal probability density function is computed at synthesis time. Taking advantage of symmetry properties, only one quadrant has been mapped into the LUT, thus achieving a reduction in the number of internal resources of the FPGA.

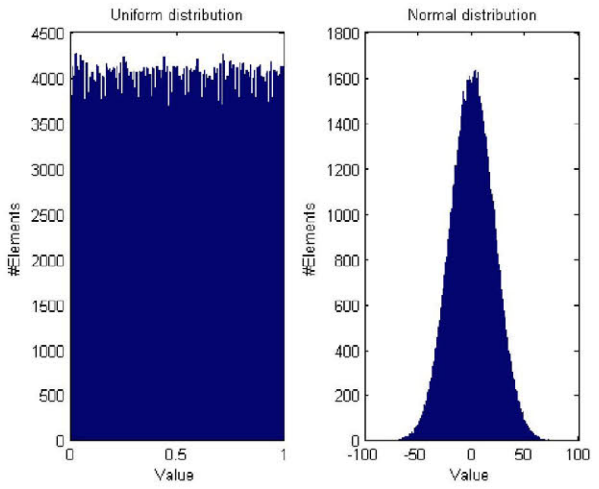

Fig. 6: Random Number Generator histograms. Uniform-distributed random numbers (left) and normal-distributed random numbers are used in different stages in the HW-EPF.

Normal distributions, on the other hand, are not that simple to obtain in hardware systems. Some implementation alternatives can be found in the literature [24], but in this work, a new strategy has been used. Using a large LFSR, twelve uniform random numbers are drawn at the same time. Then, these numbers are added, and the resulting number, as a consequence of the central limit theorem, is assumed to follow a normal distribution. This assumption is valid, as it can be seen in the distribution histograms in Fig. 6. This approach provides an accurate implementation without increasing excessively resource consumption.

5 Experimental Results

In this section, different aspects of the implemented design are put to the test. This work addresses algorithm validation, overall system functionality validation through Hardware In the Loop (HIL) co-simulation, and a sensitivity analysis of the most important parameters in the system. It also covers performance comparisons between different implementations of the same algorithm, and a summary report on resource consumption and timing rates. All those test in which an actual physical device is needed have been carried out using the XUP-V5 development board, which features a Xilinx Virtex-5 FPGA.

5.1 Evolutionary Resampling Algorithm Validation

A first set of test has been designed in order to validate the evolutionary algorithm in the resampling stage of the particle filter. Since these tests deal with the algorithm itself and not with the implementation, they have been carried out in MATLAB, not taking into account actual implementation considerations, such as fixed-point data precision or data overflow effects within the mathematical operations. The most used equations in particle-filtering performance validation tests are those from the univariate non-stationary growth model, with the addition of a quadratic measurement model. The combination of both equations provides a highly non-linear system, thus making it suitable for testing non-linear estimation capabilities. Hence, the validation equations can be expressed as (25) and (26).

$$x_k = x_{k-1} + \frac{12\, x_{k-1}}{1 + x_{k-1}^2} + 7\,\cos(1.2\,(k-1)) + w_{k-1} \quad (25)$$

$$z_k = \frac{x_k^2}{20} + \varphi_k \quad (26)$$

$w_{k-1} \sim N(0, \sigma_x)$ is the so-called process noise, whose standard deviation is represented by σ_x, and $\varphi_k \sim N(0, \sigma_z)$ is the so-called measurement noise, whose standard deviation is represented by σ_z. Both noise distributions have zero mean. Notice that the equations represent a one-dimensional system, since the aim of these tests is to validate the functionality of the evolutionary resampling stage, and not the target application itself.

Results from a comparison between the standard Bootstrap Filter and the proposed HW-EPF show that, under normal operation, both filters provide accurate estimations (see Fig. 7). However, the HW-EPF slightly outperforms, in terms of tracking performance, the Bootstrap Filter (see estimation errors in Fig. 8). Moreover, if changes are introduced in the trajectory so as to simulate an inaccurate transition process modeling, the HW-EPF is able to recover from lost-tracking states, whereas the standard Bootstrap Filter loses the tracking never to recover from that stray state. This specific condition is shown in Fig. 9.

Another important feature of the proposed resampling stage is that, as opposed to the resampling strategy adopted in the Bootstrap Filter, it does not generate sample impoverishment phenomena (at least to some extent) in the system (see Fig. 10). Furthermore, if the number of generations the evolutionary algorithm runs is not large, particle diversity is kept and thus sample impoverishment phenomena are mitigated. However, evolutionary processes in which the maximum generation limit is not set to be small might show no mitigation at all.

5.2 HIL Functional Validation

Hardware-in-the-loop co-simulation is a useful testing methodology in which the designed system is connected to a real-world environment, usually the one in which it has been designed to work. The environment is in charge of providing all necessary stimuli to the design under test and, therefore, it does not have to be modeled. Environment modeling is a highly time-consuming task in system testing or validation processes. Hence, HIL co-simulation is able to speed up the testing phase in every design process. It is for this reason that this testing methodology has been used in order to validate the right functionality of the HW-EPF, i.e. its motion estimation capabilities.

Particle filters are commonly used as the main estimation engine in many different applications. These applications include, but are not limited to, visual tracking [25] [26], object detection [27], image segmentation [28], contour detection [29], video stabilization [30], and even point set registration [31], i.e. finding a spatial transformation that aligns two given point sets. Moreover, particle filters have also been used as powerful estimation tools in other applications, such as video coding/decoding [32]. The proposed architecture can be used in almost any application scenario that requires estimations of both position and velocity of a moving object. The modular implementation ensures that the system can switch applications by simply changing the preprocessing stage. Therefore, the proposed HW-

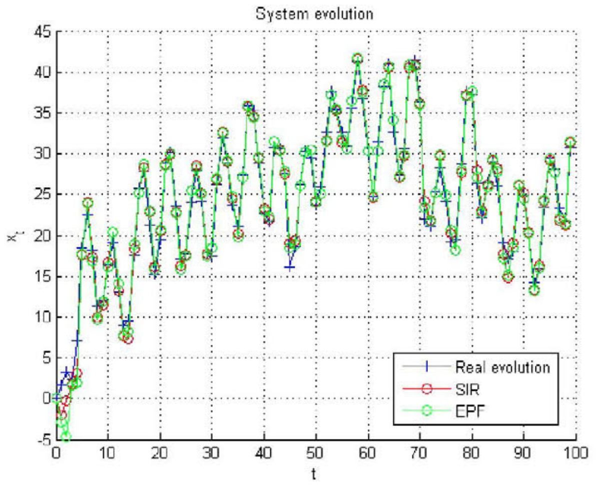

Fig. 7: HW-EPF vs. Bootstrap filter. In normal operation mode, both filters provide accurate estimations and are able to track the object. The image shows the real trajectory, as well as both filter estimations.

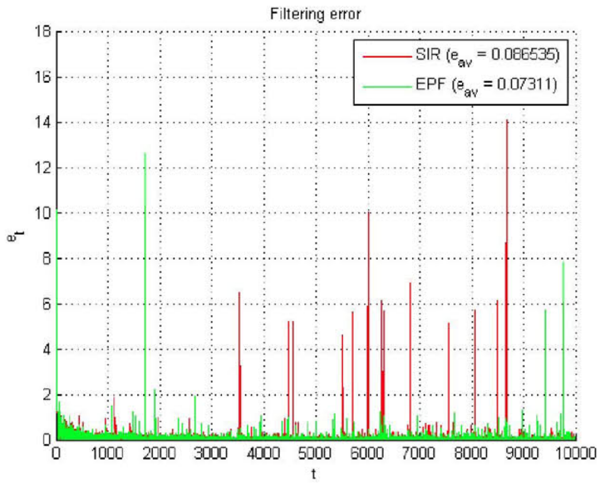

Fig. 8: HW-EPF vs. Bootstrap filter: overall estimation error. Although the tracking is lost at certain points in both filters, the HW-EPF provides estimations with less mean error. Moreover, lost-tracking situations appear more often in the Bootstrap Filter.

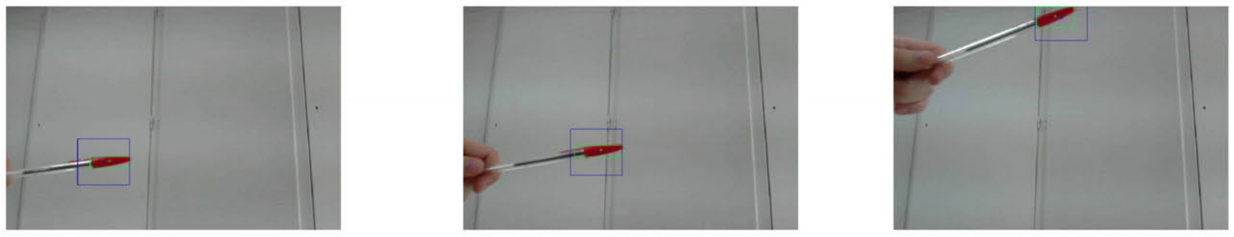

Fig. 11: HIL co-simulation results. The functional validation of the proposed architecture is carried out using a red object tracking proof-of-concept demonstrator. Results show that the HW-EPF performs estimations in real time (assuming video inputs of 30fps).

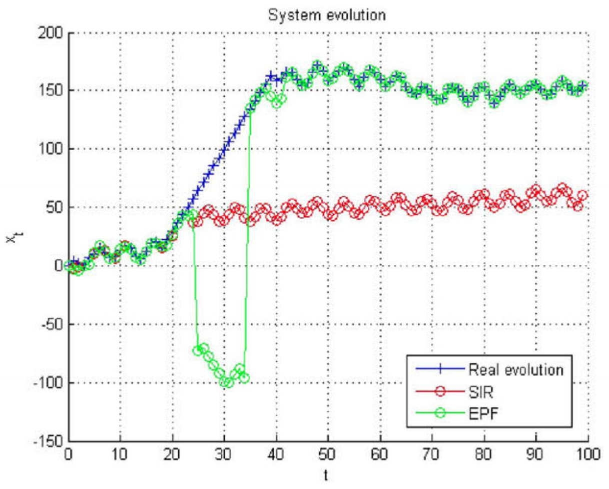

Fig. 9: HW-EPF vs. Bootstrap filter: mutation benefits. Mutation generates individuals in high-fitness areas of the state space, thus recovering from lost-tracking situations, as opposed to the Bootstrap Filter.

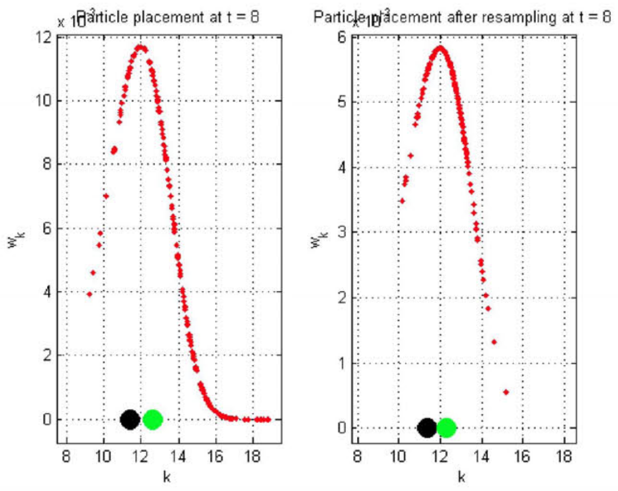

Fig. 10: Sample impoverishment mitigation in the HW-EPF. As opposed to the Bootstrap Filter, the HW-EPF assures particle diversity due to its evolutionary properties, thus providing accurate posterior distributions. These results are better if the generation count is kept at a minimum value. Notice that the population moves towards the maximum fitness point.

EPF constitutes itself a generic, application-independent motion estimation system. However, a visual tracking application has been selected as a proof of concept.

The HIL co-simulation has been set up using both Simulink and Xilinx System Generator toolboxes. Images are acquired through a webcam and then preprocessed inside the Simulink model. The target application is to track red objects within the visual space. Hence, the preprocessing stage detects those red objects, selects the biggest among them and extracts its center of mass x-axis and y-axis coordinates. These two values are the actual measured values that are sent to the evaluation board through an Ethernet point-to-point connection. The HW-EPF then estimates the position of the center of mass and sends back those estimated coordinates, which are then printed over the original image.

The obtained results prove that the HW-EPF provides an accurate tracking (i.e. estimated values), thus validating the proposed system architecture and implementation. In addition, and since normal operation times allow real-time computation (as it will be shown in the forthcoming subsections), it is possible to track objects using not only static images but also live-feed video (Fig. 11).

5.3 Sensitivity Analysis

Complex designs such as the HW-EPF require a large number of system parameters so that they can be flexible and, to some extent, reconfigurable. Therefore, it seems reasonable to analyze whether these parameters have some impact on system performance or not. Furthermore, it also seems relevant to detect on which parameter the system has stronger dependencies. The complete list of system parameters, as well as their default values can be seen in Table 3. The analysis is focused on two different, yet significant, performance variables: on the one hand, estimation error, i.e. the difference between the estimation and the actual value; on the other hand, estimation time, i.e. the time the system needs to generate a valid estimation from a valid measurement. The estimation error is evaluated as the Euclidean distance (or 2-norm distance) between both estimation and measurement points, which is shown in

TABLE 3
System Parameters: Reference Values

Parameter	Description	Value
σ_{pos}	Position standard deviation	32
σ_{vel}	Velocity standard deviation	0.01
σ_{meas}	Measurement standard deviation	10
T	Theoretical sampling time	0.033
$\#LUT\ Values$	Fitness LUT values per dimension	32
$\#Particles$	Population size	200
$\#Parents$	Parents number	10
$\#Generations$	Generations number	2
p_{cross}	Crossover probability	0.6
p_{mut}	Mutation probability	0.1
r_{mut}	Mutation ratio	0.4
σ_{mut}	Local search standard deviation	6.0

(27).

$$e_k = \sqrt{(z_{xk} - \hat{x}_k)^2 + (z_{y_k} - \hat{y}_k)^2} \tag{27}$$

z_{xk} and z_{y_k} are the measurements, whereas $\hat{x}_k$ and $\hat{y}_k$ are the estimations.

System parameters in this work can be divided into two different groups: model parameters (e.g. process noise or measurement noise standard deviations) and evolutionary algorithm parameters. The results that appear in this subsection deal with the latter.

It seems clear that population size has a huge impact on estimation performance (see Fig. 12). However, the larger the population is, the slower the estimations are and the larger the resource consumption rate is. Increasing the number of parents also leads to a very significant increase in hardware resources (more internal RAM memory is needed in both cases) and estimation times, but the estimation error is not improved (see Fig. 13). Therefore, the number of particles has to be determined by the tradeoff between accuracy, speed and area, whereas the number of parents is determined by the minimum amount that assures particle diversity and a sufficient number of mutation and crossover operations.

When the number of generations is increased, the estimation error suffers the same variation (see Fig. 14), due to sample impoverishment phenomena. Furthermore, an increase in the generation limit produces longer estimation times. Hence, the number of generations has to be kept under a certain threshold to still benefit from the evolutionary resampling stage.

Crossover and mutation probabilities, generally speaking, provide better results, in terms of estimation error, when their values are close to one. An increase in crossover probability always leads to smaller estimation errors. However, increases in mutation probability have different outcomes depending on the mutation operator that is dominant. When the mutation ratio favors random placement the estimation error is increased (see Fig. 16); when local search mutation is favored, estimation errors decreases (see Fig. 15). This reveals that, although random placement mutation is absolutely necessary in order to deal with lost-tracking situations, it must be kept at minimum values to avoid affecting system performance.

The effect that crossover and mutation probabilities have in estimation time is the opposite of that in estimation error. Larger probabilities mean more offspring particles and, therefore, more sorting operations that increase sharply estimation times (see Fig. 17). Hence, another tradeoff between estimation performance and execution time determines whether to select specific values for crossover and mutation probabilities or not.

5.4 HW vs. SW Performance Comparison

The proposed architecture targets embedded systems. The usage of hardware resources instead of software routines provides a significant improvement in terms of performance and execution time. Take for instance

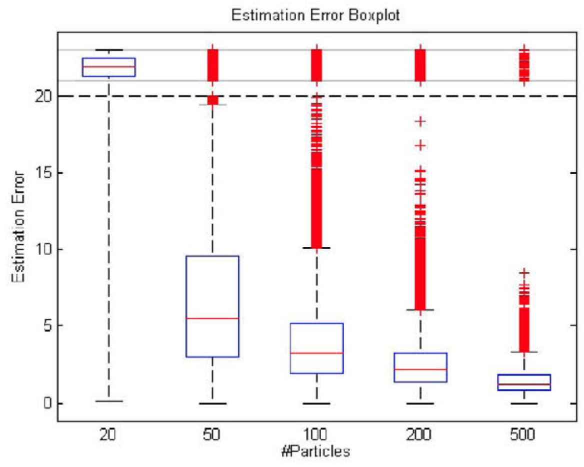

Fig. 12: Estimation error vs. population size boxplot. Estimation error can be significantly reduced if the population size is increased. Small particle population generate bad results, as it can be seen in the example of 20 particles, whereas large populations increase resource utilization.

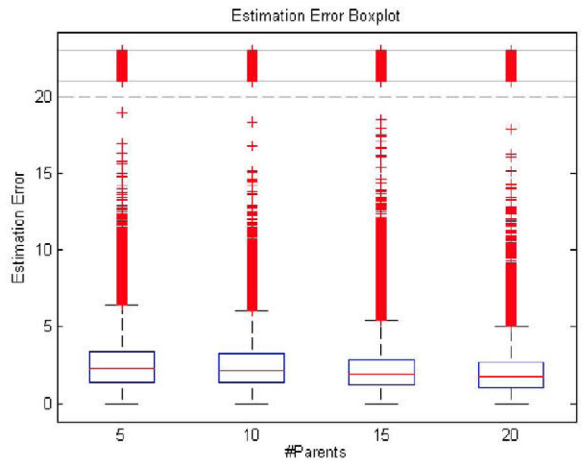

Fig. 13: Estimation error vs. parent number boxplot. Increasing the number of parents increases system performance, but only slightly. Since parent number has a deep impact on resource consumption, the fewer individuals are selected as parents, the smaller the resulting implementation is.

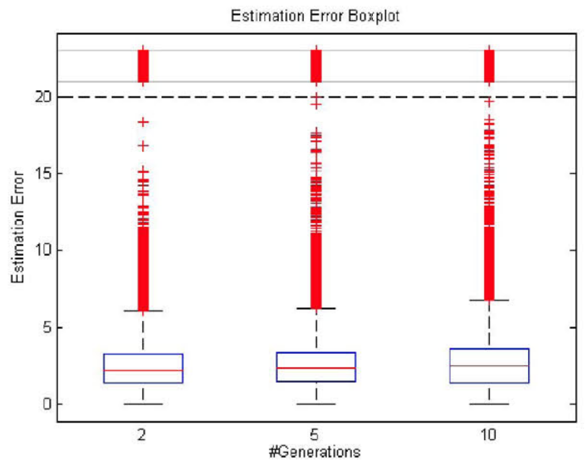

Fig. 14: Estimation error vs. generation number boxplot. Whenever the generation limit is increased, there is an increase in the estimation error, since the resulting posterior distribution suffers more sample impoverishment effects.

the fitness evaluation module, which replaces a complex mathematical equation (online computation) by a much faster indexing process (LUT addressing).

Establishing a comparison between the same algorithm implemented on different embedded platforms, it is clear that the hardware approach outperforms those that try to take advantage of software resources. Software approaches, no matter what data precision the computing core uses (fixed-point or floating-point operations), present a bottleneck in the fitness evaluation unit. Complex mathematical operations, such as exponential functions, reduce embedded systems performance. The proposed hardware accelerator performs estimations in less time that the other alternatives, independently of the number of particles that is being used. This can be seen

TABLE 4
Resource Consumption

Resource	Utilization(%)
Slice Registers	12
Slice LUTs	18
Block RAM	6
DSPs	26

Resource consumption report on the global hardware accelerator architecture. The module has been implemented in a Xilinx Virtex-5 FPGA (5vlx110tff1136-1). Notice that there is a large number of DSP processing elements used, which may generate large area overheads in other platforms.

in both Fig. 18 (minimum estimation times) and Fig. 19 (maximum estimation times).

5.5 Resource Consumption and Timing Results

This last subsection shows the resource utilization report (Table 4) and the timing results (Table 5), i.e. maximum frequency in the design. Notice that the implemented HW-EPF uses a large number of DSPs units inside the FPGA, even though the architecture has been optimized by means of resource sharing techniques. This can be considered one of the design main weaknesses. In addition, the maximum allowable frequency is not that high when compared with those of the individual modules that constitute the whole design. This is mainly due to large routes inside the FPGA. However, this maximum frequency value still provides good results when dealing with real-time video processing.

6 Conclusions

A novel particle-filtering architecture, the HW-EPF, has been designed. Its performance analysis reveals that it not only provides accurate motion state estimations, but also outperforms other algorithms, e.g. the Bootstrap

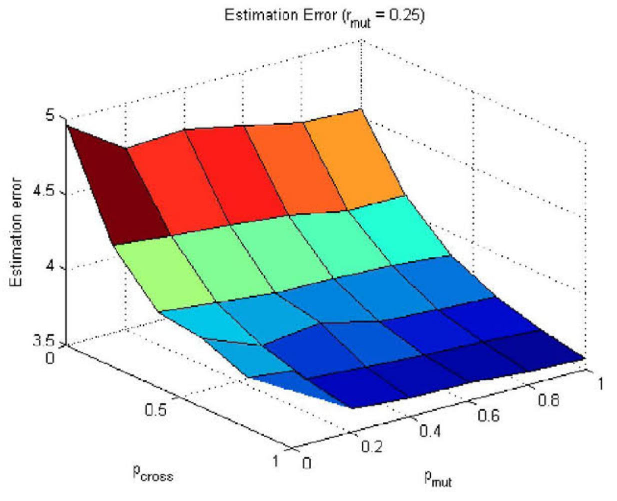

Fig. 15: Estimation error vs. genetic operators with more local search mutation than random placement mutation. Local search mutation reduces overall estimation errors. Crossover also helps reducing estimation errors, but its effects are more relevant.

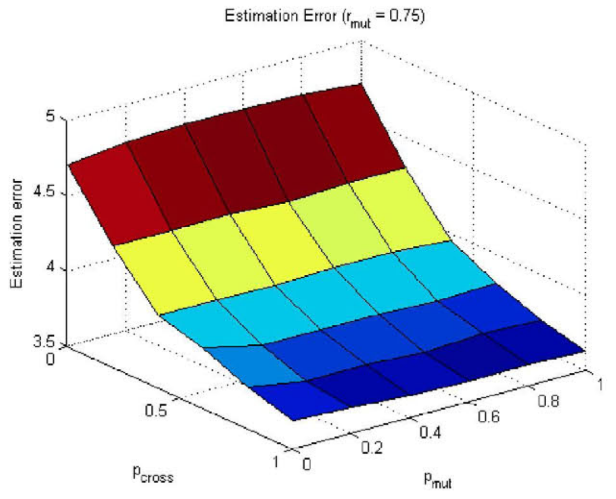

Fig. 16: Estimation error vs. genetic operators with less local search mutation than random placement mutation. Random placement mutation, although necessary in order to recover from lost-tracking events, does not help reducing estimation errors as local search mutation.

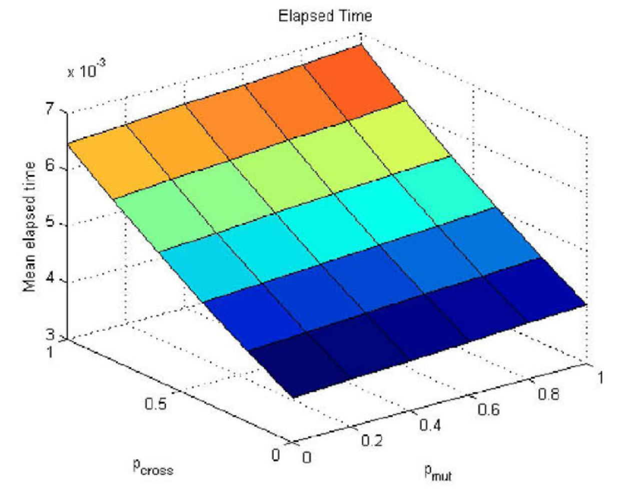

Fig. 17: Estimation time vs. genetic operators. The children count affects the time spent in each estimation. There is a bottleneck in the fitness-sorting algorithm, and its negative effects are sharper when the crossover probability is increased (two children appear, instead of only one as in mutation).

TABLE 5
Timing Report

Module	Maximum frequency (MHz)
Random Number Generator	135.080
Process model	95.716
Crossover unit	113.097
Mutation unit	99.885
Divider	227.376
HW-EPF	63.914

Timing results for each of the individual modules that constitute the HW-EPF peripheral, as well as those from the final architecture. The maximum operating frequency of the integrated design is reduced due to large interconnecting routes between modules. As in the resource consumption report, the selected device is a Xilinx Virtex-5 FPGA (5vlx110tff1136-1).

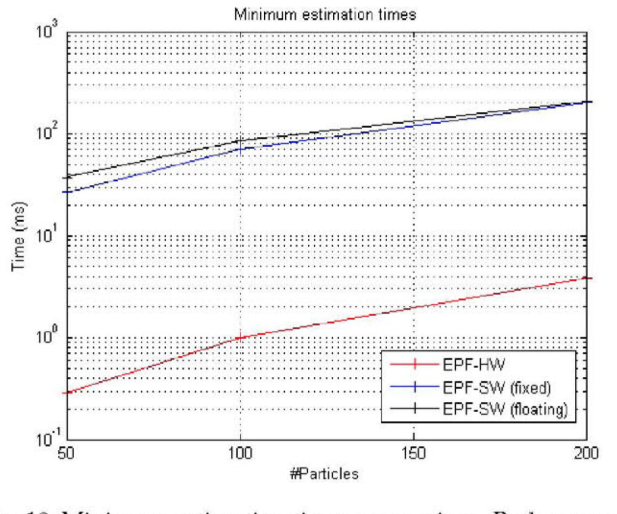

Fig. 18: Minimum estimation times comparison. Both crossover and mutation probabilities are set to zero and thus no child is generated. This is not a real situation, since there is no evolutionary influence on the estimations. However, the HW-EPF is still much faster than the other two approaches.

Filter. Particle-filtering common problems, i.e. particle degeneracy and sample impoverishment, are mitigated with the proposed algorithm, providing both accurate and realistic posterior distributions.

The HW-EPF modular architecture provides flexibility and reconfiguration capabilities to the embedded system in which is used. As a hardware accelerator, it speeds up estimation throughput. This acceleration has been verified when establishing a comparison between the same algorithm with different implementations (only hardware, software with fixed-point precision, and software with floating-point precision) over the same evaluation platform, and comes from the advantages that hardware processing has when dealing with repetitive operations.

The sensitivity analysis shows that those system parameters that increase particle number, e.g. population size, parent size (the more parents are selected, the more offspring is generated), have to remain under certain limits, in order to avoid excessively large implementations, i.e. with large area overhead. Moreover, mutation and crossover probability thresholds have to be selected taking into account the tradeoff between precision and execution time: higher values show, generally speaking, more accurate results but it takes longer to obtain the estimations. In addition, the maximum number of generations has to be small, in order to mitigate sample impoverishment phenomena and reduce estimation times.

All things considered, the HW-EPF has proved to be an outstanding filter, as well as a robust and powerful estimation tool. A proof-of-concept implementation using HIL co-simulation has been made in order to validate system functionality.

REFERENCES

[1] R. E. Kalman, "A new approach to linear filtering and prediction problems," *ASME Journal of Basic Engineering*, 1960.

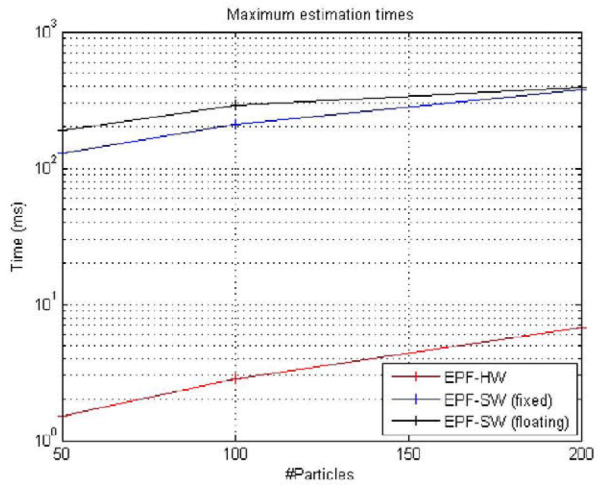

Fig. 19: Maximum estimation times comparison. Both crossover and mutation probabilities are set to one and thus all possible children are generated. Estimation times are much shorter in the HW-EPF, independently on the population size.

[2] G. Welch and G. Bishop, "An introduction to the kalman filter," Chapel Hill, NC, USA, Tech. Rep., 1995.
[3] A. Doucet and A. M. Johansen, "A tutorial on particle filtering and smoothing: fifteen years later," 2011.
[4] N. Gordon, D. Salmond, and A. Smith, "Novel approach to nonlinear/non-gaussian bayesian state estimation," *IEEE Proceedings F, Radar and Signal Processing*, vol. 140, no. 2, pp. 107–113, 1993.
[5] J. U. Cho, S.-H. Jin, X. D. Pham, J. W. Jeon, J.-E. Byun, and H. Kang, "A real-time object tracking system using a particle filter," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, Oct 2006, pp. 2822–2827.
[6] H. El-Halym, I. Mahmoud, and S. E.-D. Habib, "Efficient hardware architecture for particle filter based object tracking," in *Image Processing (ICIP), 2010 17th IEEE International Conference on*, Sept 2010, pp. 4497–4500.
[7] S.-A. Li, C.-C. Hsu, W.-L. Lin, and J.-P. Wang, "Hardware/software co-design of particle filter and its application in object tracking," in *System Science and Engineering (ICSSE), 2011 International Conference on*, June 2011, pp. 87–91.
[8] S. Agrawal, P. Engineer, R. Velmurugan, and S. Patkar, "Fpga implementation of particle filter based object tracking in video," in *Electronic System Design (ISED), 2012 International Symposium on*, Dec 2012, pp. 82–86.
[9] V. Jilkov, J. Wu, and H. Chen, "Performance comparison of gpu-accelerated particle flow and particle filters," in *Information Fusion (FUSION), 2013 16th International Conference on*, July 2013, pp. 1095–1102.
[10] M. Chitchian, A. van Amesfoort, A. Simonetto, T. Keviczky, and H. Sips, "Adapting particle filter algorithms to many-core architectures," in *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, May 2013, pp. 427–438.
[11] M. Chitchian, A. Simonetto, A. van Amesfoort, and T. Keviczky, "Distributed computation particle filters on gpu architectures for real-time control applications," *Control Systems Technology, IEEE Transactions on*, vol. 21, no. 6, pp. 2224–2238, Nov 2013.
[12] T. Higuchi, "Monte carlo filter using the genetic algorithm operators," *Journal of Statistical Computation and Simulation*, vol. 59, no. 1, pp. 1–23, 1997.
[13] B.-T. Zhang, "A bayesian framework for evolutionary computation," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, vol. 1, 1999, pp. –728 Vol. 1.
[14] S. Park, J. P. Hwang, E. Kim, and H.-J. Kang, "A new evolutionary particle filter for the prevention of sample impoverishment," *Evolutionary Computation, IEEE Transactions on*, vol. 13, no. 4, pp. 801–809, Aug 2009.
[15] C. Li, Q. Honglei, and X. Juhong, "Distributed genetic resampling particle filter," in *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, vol. 2, Aug 2010, pp. V2–32–V2–37.
[16] J. Zhang, T.-S. Pan, and J.-S. Pan, "A parallel hybrid evolutionary particle filter for nonlinear state estimation," in *Robot, Vision and Signal Processing (RVSP), 2011 First International Conference on*, Nov 2011, pp. 308–312.
[17] W. L. Khong, W. Y. Kow, Y. K. Chin, M. Y. Choong, and K. Teo, "Enhancement of particle filter resampling in vehicle tracking via genetic algorithm," in *Computer Modeling and Simulation (EMS), 2012 Sixth UKSim/AMSS European Symposium on*, Nov 2012, pp. 243–248.
[18] C. Nyirarugira and T. Y. Kim, "Adaptive evolutional strategy of particle filter for real time object tracking," in *Consumer Electronics (ICCE), 2013 IEEE International Conference on*, Jan 2013, pp. 35–36.
[19] A. Doucet, N. De Freitas, and N. Gordon, Eds., *Sequential Monte Carlo methods in practice*, 2001.
[20] A. Doucet, S. Godsill, and C. Andrieu, "On sequential monte carlo sampling methods for bayesian filtering," *Statistics and Computing*, vol. 10, no. 3, pp. 197–208, 2000.
[21] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking," *Signal Processing, IEEE Transactions on*, vol. 50, no. 2, pp. 174–188, Feb 2002.
[22] F. Desbouvries and W. Pieczynski, "Particle filtering in pairwise and triplet markov chains," in *Proceedings of the IEEE EURASIP Workshop on Nonlinear Signal and Image Processing (NSIP 2003), Grado-Gorizia*, 2003, pp. 8–11.
[23] N. Abbassi, D. Benboudjema, S. Derrode, and W. Pieczynski, "Optimal filter approximations in conditionally gaussian pairwise markov switching models," *Automatic Control, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.
[24] D.-U. Lee, J. Villasenor, W. Luk, and P. Leong, "A hardware gaussian noise generator using the box-muller method and its error analysis," *Computers, IEEE Transactions on*, vol. 55, no. 6, pp. 659–671, June 2006.
[25] Y. Zheng, Z. Shi, R. Lu, S. Hong, and X. Shen, "An efficient data-driven particle phd filter for multitarget tracking," *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 4, pp. 2318–2326, Nov 2013.
[26] J. Kwon, H. S. Lee, F. Park, and K. M. Lee, "A geometric particle filter for template-based visual tracking," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 36, no. 4, pp. 625–643, April 2014.
[27] G. Gualdi, A. Prati, and R. Cucchiara, "Multistage particle windows for fast and accurate object detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 34, no. 8, pp. 1589–1604, Aug 2012.
[28] D. Varas and F. Marques, "Region-based particle filter for video object segmentation," in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, June 2014, pp. 3470–3477.
[29] N. Widynski and M. Mignotte, "A multiscale particle filter framework for contour detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 36, no. 10, pp. 1922–1935, Oct 2014.
[30] J. Yang, D. Schonfeld, and M. Mohamed, "Robust video stabilization based on particle filter tracking of projected camera motion," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 19, no. 7, pp. 945–954, July 2009.
[31] R. Sandhu, S. Dambreville, and A. Tannenbaum, "Point set registration via particle filtering and stochastic dynamics," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 32, no. 8, pp. 1459–1473, Aug 2010.
[32] S. Wang, L. Cui, L. Stankovic, V. Stankovic, and S. Cheng, "Adaptive correlation estimation with particle filtering for distributed video coding," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 5, pp. 649–658, May 2012.

Alfonso Rodriguez was born in Madrid, Spain, in 1989. He received the BSc degree in industrial engineering and the MSc degree in industrial electronics from the Universidad Politecnica de Madrid (UPM), Spain, in 2012 and 2014.

He is currently a full-time researcher and working toward the PhD degree in industrial electronics at Centro de Electronica Industrial (CEI), UPM. His current research interests include artificial intelligence, embedded systems, high performance computing, and reconfigurable computing.

Felix Moreno (M'10) was born in Valladolid, Spain, in 1959. He received the M.Sc. and Ph.D. degrees in telecommunication engineering from Universidad Politecnica de Madrid (UPM), in 1986 and 1993, respectively.

Currently, he is Associate Professor of Electronics at UPM. His research interests are focused on evolvable hardware, high-performance reconfigurable and adaptive systems, hardware embedded intelligent architectures, and digital signal processing systems.