Instinct-Driven Dynamic Hardware Reconfiguration: Evolutionary Algorithm Optimized Compression for Autonomous Sensory Agents

Ahmed Hallawa
Chair of Integrated Signal Processing Systems
Kopernikusstraße 16
Aachen, Germany
hallawa@ice.rwth-aachen.de

Jaro De Roose
ESAT - MICAS, Microelectronics and Sensors
Kasteelpark Arenberg 10 - box 2443
Leuven, Belgium
jaro.deroose@kuleuven.be

Martin Andraud
ESAT - MICAS, Microelectronics and Sensors
Kasteelpark Arenberg 10 - box 2443
Leuven, Belgium
martin.andraud@kuleuven.be

Marian Verhelst
ESAT - MICAS, Microelectronics and Sensors
Kasteelpark Arenberg 10 - box 2443
Leuven, Belgium
Marian.Verhelst@esat.kuleuven.be

Gerd Ascheid
Chair of Integrated Signal Processing Systems
Kopernikusstraße 16
Aachen, Germany
ascheid@ice.rwth-aachen.de

ABSTRACT

Advancement in miniaturization of autonomous sensory agents can play a profound role in many applications such as the exploration of unknown environments, however, due to their miniature size, power limitations poses a severe challenge. In this paper, and inspired from biological instinctive behaviour, we introduce an instinct-driven dynamic hardware reconfiguration design scheme using evolutionary algorithms on behaviour trees. Moreover, this scheme is projected on an application scenario of autonomous sensory agents exploring an inaccessible dynamic environment. In this scenario, agent's compression behaviour -introduced as an instinct- is critical due to the limited energy available on the agents. This emphasises the role of optimization of agents resources through dynamic hardware reconfiguration. In that regard, the presented approach is demonstrated using two compression techniques: Zero-order hold and Wavelet compression. Behavioural and hardware-based power models of these techniques, integrated with behaviour trees (BT), are implemented to facilitate off-line learning of the optimum on-line behaviour, thus, facilitating dynamic reconfiguration of agents hardware.

ACM Reference format:
Ahmed Hallawa, Jaro De Roose, Martin Andraud, Marian Verhelst, and Gerd Ascheid. 2017. Instinct-Driven Dynamic Hardware Reconfiguration: Evolutionary Algorithm Optimized Compression for Autonomous Sensory Agents. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3067695.3084202

GECCO '17 Companion, Berlin, Germany

DOI: http://dx.doi.org/10.1145/3067695.3084202

1 INTRODUCTION

Evolutionary algorithms (EAs) are widely used in hardware circuit design and synthesizing different analog and digital computational circuits for wide range of applications such as controllers and filters [2] [3]. However, in many of these applications the output of the EA module is a design fitted for a specific problem and does not facilitate dynamic hardware reconfiguration. Furthermore, to facilitate on-line reconfiguration, other presented work in literature propose an intrinsic approach where the EA module is implemented on the hardware [5]. However, this is not feasible with *Miniature autonomous sensory agents* (MASA) due to hardware resources limitations. In this work we propose a scheme which uses EA to design hardware that facilitates dynamic reconfiguration in changing environments. Furthermore, this adaptive behaviour doesn't require real time complex processing or tedious learning cycles. We dub this evolved behaviour as an *instinct.* In other words, an instinct is a an adaptive behaviour learned off-line through evolutionary process. MASA offer a robust solution to many applications such as monitoring leaks and ruptures of the infrastructure such as underground water supplies, oil and gas pipes. Moreover, MASA are suitable for resource explorations oil in *cold heavy oil production with sand* (CHOPS) [8].

1.1 Problem Statement

One of the key aspects in the design of autonomous sensory agents for complex objectives such as the exploration of unknown environments, is the fact that the mathematical representation, associating different low level hardware design variables with high level defined objectives, is either not available or limited. The reason behind the unavailability is the huge design distance between such lower level variables and higher level objectives, on the other hand, limitations are caused by the existence some mathematical representation for sub-objective(s) that are independent from the other available sub-modules in the agent's hardware architecture. This leads to the first design objective:

Objective 1.1. *The learning process of the behaviour must rely primarily on the behaviour performance metrics, i.e. its fitness, and on the assumption that only limited, if any, access to the mathematical formulation relating available tunable variables with the defined objectives is available.*

Empirically, it is assumed that it is infeasible to know a priori which agent will be where in the environment and when, at least with reliable probabilistic certainty. Therefore, the learned behaviour shall be used with all agents and must function effectively everywhere in the environment:

Objective 1.2. *The evolved behaviour must be universal, i.e. fit to be used by any agent at any point on the spatial and temporal dimensions.*

Moreover, this evolved behaviour will be implemented in hardware, consequently, it must be integrable with it. Hence, even if its actions optimize the use of the available hardware to achieve user's defined objectives, the behaviour's control flow itself triggering these actions must also be efficient and should minimize the use of hardware resources:

Objective 1.3. *The learning process of the agent's behaviour must attempt to find the control flow that achieves its functionality with least possible consumption of hardware resources.*

1.2 Biological Instincts

In biological systems, any behaviour is considered instinctive if it is performed without being based upon prior self experience but rather on hereditary bases. This means that the learning process is done on the genetic level, i.e. the learning experience is accumulated genetically. Such learned behaviours found in nature may include a simple, low level and fast or a complex and multi-system reaction(s) to a given stimulus.

This instinctive behaviour is of exceptional interest to presented problem statement, as mimicking it can play an important role in the behaviour learning scheme of the agents. On one hand, instinctive behaviour is genetic based, thus, learning is conducted on the genetic level, which gives the power to achieve solving a problem without — or with very limited — understanding to its mathematical formulation and only based on its performance on that environment (Objective 1.1). Furthermore, in many species this behaviour is fast, reflex-like, and doesn't require extensive computation to be conducted, which is particularly suited for our problem due to the agent's resource limitations (Objective 1.3). For all these reasons and more, biological instincts are adopted as our design methodology, and the presented design in this paper is an attempt to mimic it.

For the rest of the paper, instinct will refer to the agents' behaviour. One of the first challenges in adopting the instinct concept is how to represent this instinct in a way that fits the problem statement and achieve its defined objectives. In the next section, a representation method called *Behaviour Trees* (BTs) is introduced.

1.3 Behaviour Trees

As an alternative to FSM, BTs were introduced in the gaming industry mainly to offer a more modular, flexible and readable representation scheme. And because of these reasons, many famous games now use it for developing their AI. In addition, they have become widely popular in robotics and unmanned air vehicles.

Formally, a BT is depth-first acyclic directed graph. Any node in this tree is either a parent, i.e. connected to lower level nodes dubbed as a *child* nodes, or a *leaf* node which has no child nodes. Moreover, each BT has one unique parent node named *root* which can not have a parent and has only a single child.

Each node can have one of three possible states: Success, failure or running. Furthermore, all nodes, except the root node, fall under two main categories: control flow nodes or execution (leaf) nodes. Control flow nodes are further sub-categorised to: composites nodes and decorative nodes, each include wide range of possibilities such as selector and parallel nodes. On the other hand, the execution (leaf) nodes have only two possibilities: action or condition nodes. The most commonly used nodes and their functionality can be described as follows:

- **Leaf Nodes**
  - **Action node**: Returns *running* when the given action is still running, *success* when it is done and *failure* otherwise.
  - **Condition node**: Returns *success* when the condition is fulfilled and *failure* otherwise.
- **Composite Nodes**
  - **Selector node**: Sequentially ticks its children nodes, starting from the out-most left node, and returns the state of the first non-failing child, i.e. either *success* or *running*. Otherwise, it returns *failure*.
  - **Sequence node**: Sequentially ticks its children nodes, starting from the out-most left node, and returns the state of the first non-succeeding child, i.e. either *failure* or *running*. Otherwise, it returns *success*.

As for the execution process of BTs, it starts from the root *ticking* its only child node, which consequently start ticking its children signalling the permission to start their functional properties, and, consequently, return their respective state to the parent that ticked them. Figure 1 shows a BT example without the root node.

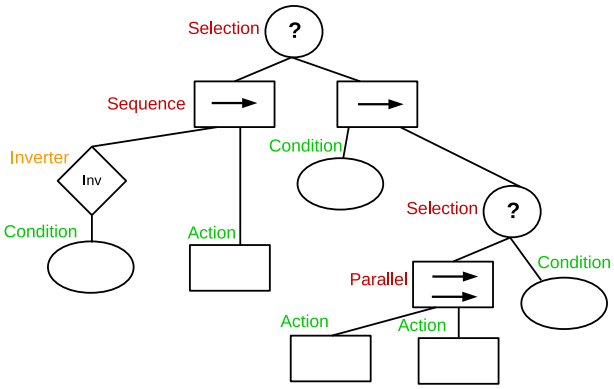

Figure 1: BT: an example

Given this description, it important to highlight why BT is an appropriate scheme for representing instincts. Firstly, adding and removing nodes can be easily done without any changes in either their parent nodes or child nodes. Furthermore, since it is

an acyclic graph with a root node, it facilitates hierarchical expansion. Moreover, the availability of dedicated nodes for actions and conditions gives the user design flexibly. And since it is a tree by structure, its complexity in representation can be a quantitatively measured, which is in agreement with Objective 1.3. One important aspect is handling parallelism, with the ticking method, all deadlock avoidance algorithms are not needed, thus minimizing complexity which is reflected positively on hardware resources.

Finally, having a dedicated nodes for actions and conditions gives the user significant flexibility and gives any learning module a wide design space. The challenge now is how to integrate a BT in the learning mechanism of an EA module?

1.4 Grammatical Evolution

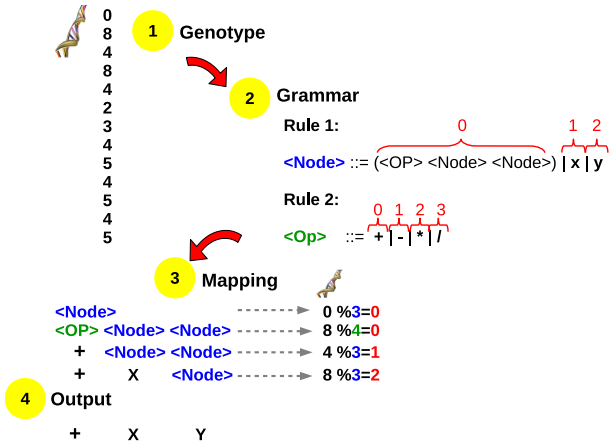

Figure 2: Grammatical Evolution: Example

One of the most common concepts used in Computer Science is the concept of *grammars* [1, 6]. Like in non-programming languages, their main objective is to restrict a certain domain via the definition of possible legal expressions.

In grammar terminology, a non-terminal character is a character that can be replaced with other characters to eventually turn into a terminal one using the production rules, on the other hand, a terminal character is an elementary character that can not be further replaced.

In EA context, grammar plays an important role in the genotype-phenotype mapping process [4] [7]. Conceptually, the genotype-phenotype mapping is the interpretation of a genotype of an individual in the population to an actual solution. In that regard, if the solution is a structured entity that can be defined with a set of rules, grammar then can be used for mapping and to guarantee that the generated interpretation is syntactically correct.

Figure 2 illustrates an example of using a grammar in genotype-phenotype mapping, the idea is to map the genotype given in step 1 to an output math operation in the prefix format, e.g + y x, from a starting non terminal given character <Node>. To do this mapping, a grammar (rules are written in Backus Naur Form (BNF)) is given with two rules. Rule 1 states that the non-terminal character <Node> can be replaced with one of three possibilities. Similarly, rule 2 states that the non-terminal character <Op> can be replaced with one of 4 possible terminal characters: +, -, *, and /. Now, let us use the grammar to know which operation the genotype will replace <Node> with: The first genotype value is 0, and since the code started a with the non-terminal character <Node>, therefore rule 1 is applied and since rule 1 has 3 possibilities, mod 3 will be used on the genotype value, leading to 0 mod 3 = 0, therefore, <Node> becomes <OP> <Node> <Node>. Similarly, after reading genotype values 8, 4 and 8, the final output will be + x y. This methodology can be used in the genotype-phenotype mapping of BTs, facilitating its evolution. Next, the full instinct evolution scheme is laid out.

2 INSTINCT EVOLUTION SCHEME

So far the objectives regarding agent's behaviour and its learning process is identified, the general methodology is chosen: mimicking biological instinctive behaviour, the representation scheme of an instinct is picked: behaviour trees and its genotype-phenotype mapping method is chosen: grammatical evolution. Now a full description to the instinct evolution scheme is presented. The scheme has six main steps as shown in Figure 3.

To give an overview on the instinct evolution scheme, a bottom-up approach will be used starting from step 6: In step 6, the evolution of the BT using grammatical evolution is conducted, the objective here is to allow the EA process to explore the search space efficiently and generate the optimum behaviour in the form of instruction set. As described earlier, BTs are flexible and facilitate user defined actions and conditions. Additionally, there is a wide range of possible nodes, each facilitate different functionality. Consequently, defining pools of possible *interesting* actions, conditions and node types before the commencement of the evolution process will achieve multiple objectives: It will minimize the solution space and, consequently, facilitate faster convergence in producing the optimum behaviour. And although faster convergence is figure of merit in any optimization scheme, it is of an exceptional importance as the simulation process of environment is computationally exhaustive. Furthermore, it will allow to use the scheme on different levels on the hardware architecture without changing its structure.

As a result, steps 3, 4 and 5 are designed for that purpose. However, in order to define *interesting* actions, an optimization on the possible tunable variables relative to user defined objective must be conducted first, which is done in step 2. Finally, to pin point all possible tunable variables for each sub-module in the hardware architecture to conduct step 2 and to identify possible perception points in the hardware needed in step 5, an agent abstraction is needed to highlight all these elements, this is executed in step 1. For further analysis to each introduced step, a detailed description is given in the following:

2.0.1 Agent Abstraction. Motivated by the definition of action and condition pools, the agent is abstracted into three main layers: Layout layer, configuration layer and perception layer

These layers are designed in order to be able to explicitly identify all possible configurations and their respective hierarchical

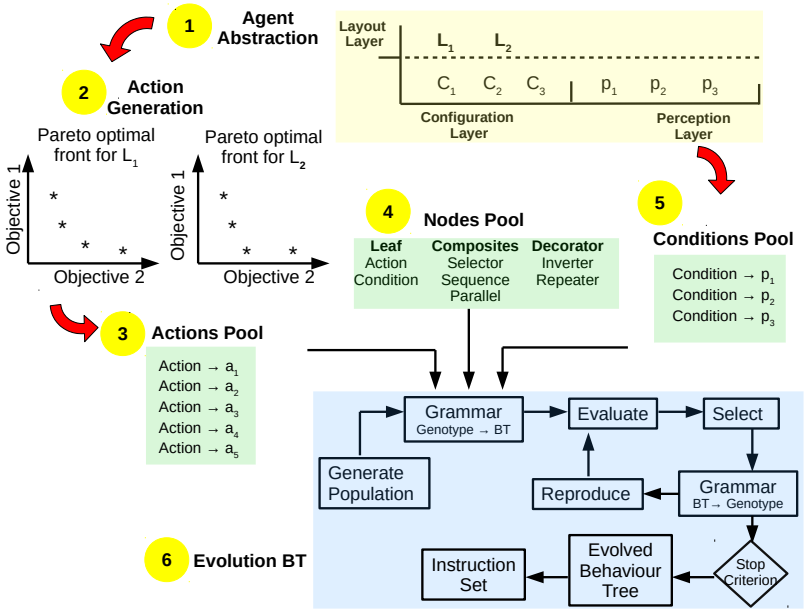

Figure 3: Instinct evolution scheme

position. Identifying the relationship between possible configurations and hierarchy will allow the EA process to control effectively the evolution process with the fitness function, e.g. setting a higher cost in the configuration change that is done on a higher hierarchical level relative to a lower one.

In the layout layer the each configurable module in the hardware architecture identifies all possible sub-modules it can activate. e.g. the compression module might have three possible sub-modules to run: a Zero-order hold (ZOH), a wavelet and a Fast Fourier Transform (FFT). In the layout layer these sub-modules are identified.

However, the configuration layer contains the set of configurable parameters for each of these sub-modules.

Both the layout layer and the configuration layer are linked to the actions pool. But, in order to identify the points where the conditions can be applied, a perception layer is used, where all sensing points interacting with the environment are identified, e.g. temperature and pressure sensors.

2.0.2 Action Generation. In this step a multi-objective algorithm is used define the Pareto optimal front of the variables in the configuration layer for each sub-system in the layout layer relative to user defined objectives. It is suggested to use non-dominated sorting genetic algorithm. In this process the user is given a chance to identify the objectives relevant to sub-module under investigation.

The outcome of this process is a set of *interesting* actions (hardware configurations) for each element in the layout layer, which will help in the generation of actions pool. It is worth mentioning that setting the maximum genotype integers will lead to setting the max number an action ID can have, which will finally set the max number of actions a pool can take.

2.0.3 Actions Pool. The set of configurations extracted from the Pareto-optimal front constitute a pool of potential actions an agent can conduct, setting up the actions pool. Each action is identified by: A unique ID to the sub-module(s) the action is associated to (layout layer), a configuration ID with a set of the the size of configurable variables with the respective configurations. (configuration layer)

A pre-set number of integers will be allocated to be used in identify the action from the genotype value as explained in details in section 2.0.6, this number will set the maximum possible actions available in the action pool.

2.0.4 Nodes Pool. In this step the pool of all possible control flow nodes of interest are defined. Each type of node has a different complexity in their implementation in the instruction set, and like all other pools, the trade-off between extending the solution space to include more possible nodes versus flexibility by having a wider range of nodes, each adding a new dimension of functionality, must be analyzed and controlled in the fitness function.

Furthermore, although BTs come with a huge list of possible control flow nodes and possible extensions, e.g the probability extension where the parent node chooses with a certain probability which child node to tick, it is also possible to define new nodes to fit design needs.

2.0.5 Conditions Pool. All entities of the perception layer represent a potential condition point to be monitored for further triggering of an action, and as a result all these points are fed to the condition pool.

Each entity in the condition pool has three basic dimensions: A unique ID to its real perception point, for example an accelerometer can read three axis, therefore, the possible IDs are three, number of threshold points to be applied on sensor readings, e.g. one

threshold means classify sensor readings into two segments. Finally, the respective position of the threshold point(s) within the sensor range, i.e. where is the threshold(s) identified in the previous step is placed within the sensor range.

It is important to highlight here, consequently, all definition of threshold points and their positions must be conducted in probabilistic manner, i.e multiple runs and from multiple agents. And with each conducted experiment, this probabilistic knowledge is accumulated converging to the best threshold definitions and for all agents as a whole, not for a specific agent in a specific run.

2.0.6 Evolution of Instincts. At this point we have three different pools action pool, condition pool and node pool. The first step is the generation of a population, then a grammatical evolutionary process is executed with the following BNR rules

$$< BT > ::=< T, T > \quad (1)$$

$$< T > ::=< T, T > | < N > | < A > | < C > \quad (2)$$

$$< N > ::= n_1 | \ldots | n_{i-1} | n_i \quad (3)$$

$$< A > ::= a_1 | \ldots | a_{j-1} | a_j \quad (4)$$

$$< C > ::= p_1 | \ldots | p_{k-1} | p_k \quad (5)$$

Where BT is the start symbol, T sets a non-terminal character, N sets the grammar rule for choosing a node n_i from the nodes pool, A sets the grammar rule for choosing an action a_i from the actions pool, C sets the grammar rule for choosing a condition p_i from the conditions pool. It is important to highlight that this used grammar can be easily extended.

3 APPLICATION SCENARIO

In order to project the introduced instinct evolution scheme on exploration of unknown environment case, a scenario is introduced Figure 4. In this scenario, agents are injecting in a water loop. In this context, compression is critical due to the limited energy available on the agents, this brings emphasis the role optimization of agents resources through dynamic hardware reconfiguration (online behaviour). Ideally, the agent should use a compression technique with low power and high information loss in the environment zones where the changes are not significant (green zone) and use another configuration in the (red) zone, where it invests more energy to capture this highly dynamic zone properties.

In this example, the hardware architecture offers two compression techniques: Zero-order hold and wavelet and the objective is develop an instinct off-line using EA that will make the agent capable of capturing the environment properties in different environment zones with the least possible power consumption by reconfiguring its available tunable parameters.

3.1 Experiment setup

One end of the pipe loop was connected to an industrial tank directly. The other end of the pipe loop was 'connected' to the tank via a flexible hose of the same diameter. Pipe loop was roughly 60 m long (30 m and 30 m) and 10 cm in diameter.

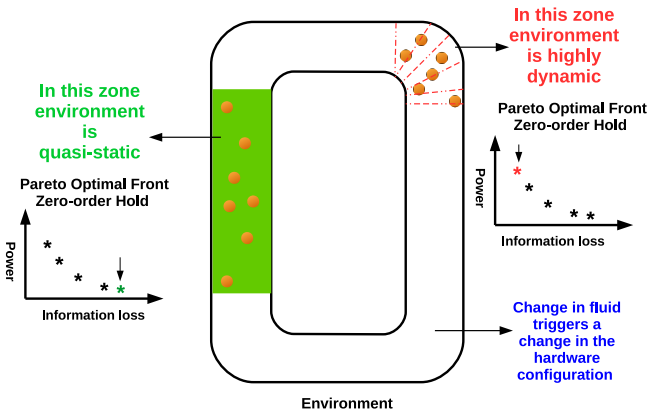

Figure 4: Sensory agents in different environment zones

3.2 Agents

Each agent has three sensors, a gyroscope, an accelerometer and a magnetometer. Each sensor has an x, y and z-axis which each produce a data stream, so that there are 9 data streams in total. In the end, we are interested in knowing the acceleration of the agent in the length, width and height directing of the pipe. However, most of the time, these directions are not the same as the x, y and z-axis of the agent, since the agent often spins. In order to match these two coordinate systems, we make use of the magnetometer. Since there usually isn't any special magnetic field present around the experiment, the magnetometer only measures the magnetic field of the earth, which always points in the same direction in respect to the pipe orientation. By measuring the magnetic field of the earth inside the coordinate system of the agent, and knowing what it is in the coordinate system of the earth, we can calculate the relation between the two. We are now able to calculate the acceleration in the coordinate system of the earth, starting from the acceleration in the coordinate system of the agent.

4 HARDWARE ARCHITECTURE

This section will describe the hardware architecture of the targeted reconfigurable sensory agents. As shown in Figure 5, the sensory subsystem of these agents consists of seven main blocks: sensors, an ADC, a buffer to hold samples, a parameter calculator, an instinct, an arithmetic unit for compression and a non-volatile memory for data storage. This section will especially focus on the reconfigurability knobs of this subsystem, which will subsequently be exploited by the evolutionary algorithm for optimal run-time operation. Moreover, we will describe the hardware implementation of the digital subblocks, the arithmetic unit and the data storage in more detail. We end this section by elaborating on the link between the simulated reconfigurable hardware and the software model of this hardware, used by the evolutionary algorithm.

4.1 General structure

Figure 5 shows the hardware structure of the agent's sensory system. It starts with a sensor that measures a physical entity and converts it to an analog signal. Targeted sensors are a.o. accelerator, pressure, or temperature sensors. The analog data from the

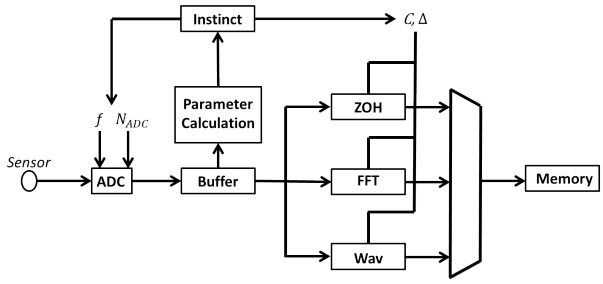

Figure 5: Overview of the hardware architecture

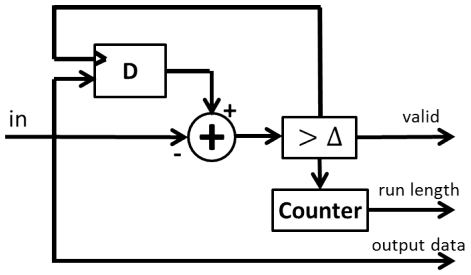

Figure 6: The structure of the zero order hold hardware

sensor is converted to a digital signal with a sampling frequency f and resolution N_{ADC} [A 0.20 mm^2 3nW Signal Acquisition IC for Miniature Sensor Nodes in 65 nm CMOS]. The samples of the ADC are stored in the buffer. Once the buffer is filled with samples, it releases all its samples at a high speed to the parameter calculator. Once the paramaters are known, these are sent to the instinct, which then puts the settings for the digital compression. The buffer sends all its samples for a second time, this time to the digital compression block. These samples are then compressed with one of the digital compression algorithms (C) and with a tolerable compression loss Δ, which will be explained in more detail later. This compressed signal is finally stored in a non-volatile Flash memory. Both the compression algorithm and Flash memory are simulated in a 90nm TSMC technology.

4.2 Compression

The sensory agent can make use of one of 3 different lossy compression algorithms, for efficiently storing its captured sensory waveforms. All three compression algorithms have been implemented in hardware: zero order hold, Fast Fourier and stream wavelet compression. As will become apparent from the results presented in this paper, it is important to enable a variety of compression mechanisms in the sensory agent. Depending on the characteristics of data, different algorithms deliver a favorable compression vs information loss trade-off. At all times, only one compression algorithm is active, while the two other hardware compression blocks are in sleep mode, powered down to reduce their leakage power. A MUX decides which set of outputs of the three compression algorithms should be stored in the memory, based on the control knob C. Following paragraphs will describe in more detail the internal structure of each of the three compression blocks, and highlight their tunability aspects.

4.2.1 Zero order hold compression. The zero order hold compression operates by comparing the current sample with the last stored sample, as shown in figure 6. If the absolute value of the difference is smaller then Δ, no new data is stored. Only if the absolute value of the difference between these two is bigger then Δ, the current value is stored, both locally for further comparison, as well as in the non-volatile storage. For the latter, also the amount of clock cycles since the last stored value is recorded, which is calculated by a counter. This counter is reset each time a new sample is stored. This block requires very little hardware: only one adder, a comparator, a MUX, a delay element and a small counter.

4.2.2 Streamed wavelet compression. Wavelet compression filters the input data using wavelets. The complete filter has 6 stages in this implementation, and is shown in figure 7. Each stage splits the input data into a high frequency and low frequency band using a low pass (LP) and high pass (HP) filter respectively and then decimates both outputs by a factor 2. The output of the low pass is then used as input for the next stage and the output of the high pass filter is sent to the output. These 4 tap low and high pass filters are built from the Daubechies scaling and wavelet function. We choose this wavelet type and number of taps for its good compression vs calculation complexity trade-off.

Compared to traditional packet-based wavelet transforms, this wavelet transform operates in a stream-based way. By using a set of filters, which continuously operate on the incoming data stream, a slightly better compression is achieved, while implementation complexity can be reduced. The stream based approach requires less registers, multipliers and adders, which is beneficial in this highly leakage dominated system. The output consists of 6 high pass outputs and one low pass output, which all have different update rates due to the internal down-sampling as shown by figure 7. By combining all the outputs in an organized fashion, exactly one output value is produced every clock cycle. This makes it easier to pipeline the system, and hence operate at low frequency with less hardware to further reduce power consumption wasted on leakage current. This one output line is compared with the loss factor Δ, to decide upon storage of the wavelet result. Only output coefficients with an absolute value larger than Δ are stored into non-volatile memory, along with the number of clock cycles since the last stored output.

4.3 Data storage

The data storage is done in an on-chip 90nm TSMC Flash memory. The application needs non-volatile memory, since the agent could run out of energy, but the data should be retained. Also, the leakage of non-volatile memories is usually too high. Our memory has a leakage of about 160nW for 32,7kB. More blocks will be needed, but only one block is active at a time. The leakage can be further reduced by employing a rush-to-sleep scheme: buffering the output of the compression in volatile memory and power gating the Flash memory while the buffer is filling. Once the buffer is full, the Flash memory is turned on, the data in the buffer is being written into the Flash memory. When the buffer is empty again, the Flash memory is again powered down. The energy to write one data word of 16 bits to the memory is 178nJ, which for our application

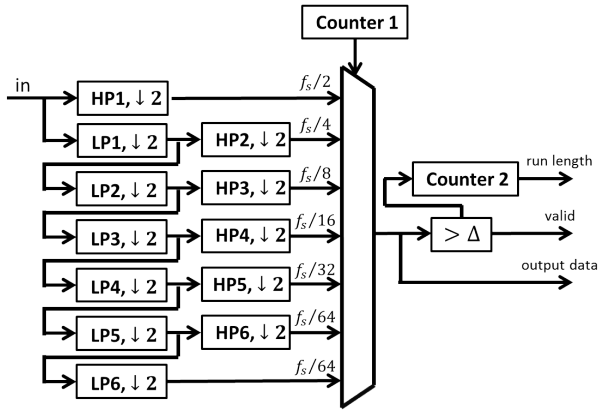

Figure 7: Structure of a wavelet filter bank

is the dominant source of energy consumption.

4.4 Behavioral model of the hardware

The behavioral model of the compression block is capable to simulate the compression operation of a given data stream, in function of the various tuning knobs N_{ADC}, f, C and Δ. This allows to quickly assess the information loss on the sensory data in function of the compression operation tuning knobs. This information loss is defined as the Percentage Root-mean-square Difference (PRD), being:

$$PRD = \frac{\sqrt{\sum_{n=1}^{k}(X_n - Y_n)^2/N}}{X_{max}} = \frac{RMS(ErrorSignal)}{MaximalError} \quad (6)$$

In which X is the original sensory signal, and Y is the reconstructed signal which can be derived from the compressed data stream, using perfect reconstruction, as the decompression is done offline.

The behavioral model is co-simulated with the extracted hardware model (see next subsection) in order to speed up the calculations required for the evolutionary algorithm, and to have all calculations originating from a common environment.

4.5 Power consumption model of the hardware

The evolutionary algorithm needs to be able to rapidly assess the impact of tuning knob settings on the system's total power consumption. This assessment would be too slow when putting hardware simulations in the loop. to speed this up an accurate energy model has been constructed. This energy model is derived from extensive simulations of the 90nm CMOS hardware implementations. The model is moreover parametrized in function of the different tunability knobs N_{ADC}, f, C and Δ, to quickly assess their impact.

The complete power consumption of the system equals

$$P_{Total} = P_{AFE} + P_{ADC} + P_{BUFF} + P_{DSP} + P_{MEM} \quad (7)$$

P_{AFE} is the power consumption of the analog front end, P_{ADC} is the power consumption of the analog to digital converter, P_{BUFF} is the power consumption of the buffer. P_{DSP} is the power consumption of the digital signal processing, this entails the Parameter Calculation (PC) block, the Instinct block, the Wavelet compression block and the Zero Order Hold compression block. P_{MEM} is the power consumption of the memory.

Operation	P_{active}	P_{leak}
P_{AFE}	+-1nW	/
P_{ADC}	$0.878{*}4^{(bits-10)}$pJ*f	0.15nW
P_{BUFF}	6.91pJ*f	611nW
P_{DSP-PC}	$(0.745bits^2$-$6.71bits$+317)fJ*f	1.1nW
$P_{DSP-Inst}$	0.09pJ*f	0.07nW
$P_{DSP-ZOH}$	$(0.29{*}f+100/cr){*}(0.01bits^2$-$0.14bits$+1.37)pW	0.14nW
$P_{DSP-Wav}$	$(16.6+1.4{*}bits)$pJ*f	14nW
P_{MEM}	11.1nJ/bit written	160nW

Table 1: Frequency of Special Characters

Table 1 shows the used active and leakage power of each operation. These results are extraced from synthesis simulations in 90nm TSMC and datasheets. All DSP units that calculate, have a square relation with the amount of bits in a sample for the active power consumption. The ZOH calculation power is also depending on the variability of the data, which is modeled with the compression ratio (cr). The compression ratio equals the total amount of samples divided by the amount of saved samples. When the signal is highly variable, the compression ratio decreases, and the power consumption increases.

5 RESULTS

Projecting the instinct evolution scheme is summarized as follows: In the agent abstraction step (step 1), the layout layer identifies two compression sub-modules: ZOH and wavelet. Each has three configurable variables setting the configuration layer. In step 2, using non-dominated sorting genetic (NSAG-2) algorithm, the Pareto optimal front of the recognized variables in the configuration layer is generated with the two objectives: information loss and agent power, this is done separately for the two techniques identified in the layout layer.

Assuming a given pre-set maximum size of the genotype of the instinct BT (50 integers), the maximum allocated number of bits representing actions, conditions and nodes can be set.Therefore, solutions are selected from the Pareto-optimal front. For the nodes pool a simple *selection* and *sequence* node types are used. In step 5, since the perception layer identifies 6 points and the number of threshold possibilities is set as 3 and assigned to 1, 2 and 3 the total number of conditions is 18. Statistical analysis done on the data available from real world experiment identifies the upper and lower limit for each condition and the position of thresholds (only single value for each threshold was used).

In 5.1 the optimization output, which constitutes the action pool, from the optimizing the compression model using NSGA2 is presented. Moreover, in 5.2 the evolutionary process of the BT settings and convergence graph is given. Finally, in 5.3, we present the hardware model verification tests results.

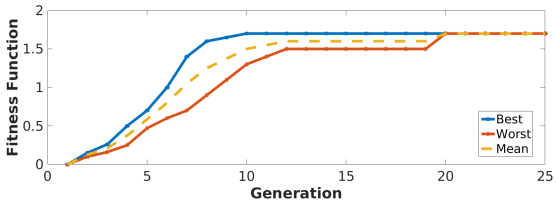

Figure 8: EA convergence

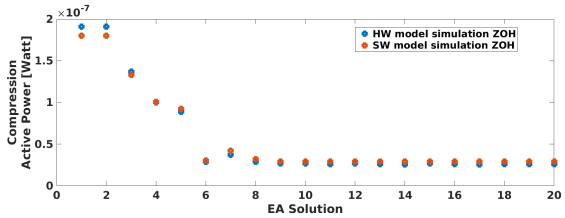

Figure 9: HW-SW models energy output for zero order hold module

5.1 Action Pool

In this subsection we present the Pareto optimal front provided from the NSGA2 module constituting the actions pool. Table 2 highlights selected solutions on the Pareto-optimal front with their relative solution configuration, where Δ is tolerable error of compression in bits, F_s is the ADC sampling frequency in Hz and *Bits* are its resolution.

Δ [#]	F_s [Hz]	Bits [#]	Power [W]	Information Loss [%]
49	487.2376	10	7.82E-06	0.9321E-03
161	159.9791	9	2.64E-06	2.7078E-03
197	151.0169	9	2.07E-06	3.4735E-03
4263	100.8162	8	4.89E-07	37.9840E-03
16916	100.0042	8	4.67E-07	63.2073E-03

Table 2: Selected Pareto optimal solutions for ZOH

5.2 Evolutionary Behaviour Tree

Using a population of 50, mutation rate = 0.03, crossover rate = 0.5 and for 20 iterations, the cost function is set as follows:

$$\Sigma = A + \eta C \tag{8}$$

Where η is a weighting coefficient, A is the environment property identification probability and C is the relative complexity defined as follows:

$$A = Pr(\text{Agents identifying correctly environment zones}) \tag{9}$$

$$C = 1 - \frac{\text{number of BT nodes}}{\text{max number of BT nodes}} \tag{10}$$

Consequently, $A \in [0,1]$ and $C \in [0,1]$. Figure 8 illustrates the convergence graph of the BT for all generations

5.3 Hardware model verification

To verify our model of the hardware, we re-evaluated the solutions from the Pareto optimal front generated with the model via the NSGA2 module for comparison and further analysis. Figure 9 illustrates the differences between the hardware simulations and the hardware model for zero order hold compression for a few solutions which are picked over the complete length of the Pareto optimal front.

6 CONCLUSION

In this paper different reasons for mimicking the biological instinctive behaviour to facilitate dynamic hardware reconfiguration in MASA were presented. This was done by defining the main design objectives of agent's behaviour and its learning process. A proposed scheme for representing instincts was laid out (BT) and a genotype to phenotype mapping methodology was illustrated. All this led to the introduction of an instinct evolution scheme using EAs (Grammatical Evolution). Furthermore, the presented instinct evolution scheme does not require any mathematical formalization between tunable hardware variables and user defined objectives. And although achieving such objective usually requires significant computation resources since the search space tends to expand, the scheme structure offers a solution to that by the minimization of the solution space via introducing actions pool, conditions pool and nodes pool to BT. Additionally, this allows user defined actions, and node types to be introduced in the solution space.

7 ACKNOWLEDGEMENTS

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 665347.
We acknowledge the computational resources provided by RWTH Compute Cluster from RWTH Aachen University under project RWTH0118.

REFERENCES

[1] Alan Bundy and Lincoln Wallen. 1984. *Context-Free Grammar.* Springer Berlin Heidelberg, Berlin, Heidelberg, 22–23. DOI:http://dx.doi.org/10.1007/978-3-642-96868-6_41
[2] Dipankar Dasgupta and Zbigniew Michalewicz. 2013. *Evolutionary algorithms in engineering applications.* Springer Science & Business Media.
[3] Peter J Fleming and Robin C Purshouse. 2002. Evolutionary algorithms in control systems engineering: a survey. *Control engineering practice* 10, 11 (2002), 1223–1241.
[4] Maarten Keijzer, Michael O'Neill, Conor Ryan, and Mike Cattolico. 2002. Grammatical evolution rules: The mod and the bucket rule. In *European Conference on Genetic Programming.* Springer, 123–130.
[5] Jörg Langeheine, Joachim Becker, Simon Folling, Karlheinz Meier, and Johannes Schemmel. 2001. A CMOS FPTA chip for intrinsic hardware evolution of analog electronic circuits. In *Evolvable Hardware, 2001. Proceedings. The Third NASA/DoD Workshop on.* IEEE, 172–175.
[6] Robert I Mckay, Nguyen Xuan Hoai, Peter Alexander Whigham, Yin Shan, and Michael O'neill. 2010. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines* 11, 3-4 (2010), 365–396.
[7] Conor Ryan, JJ Collins, and Michael O Neill. 1998. Grammatical evolution: Evolving programs for an arbitrary language. In *European Conference on Genetic Programming.* Springer, 83–96.
[8] Stephan Schlupkothen and Gerd Ascheid. 2016. Joint Localization and Transmit-Ambiguity Resolution for Ultra-Low Energy Wireless Sensors. In *2016 IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE'16).*