Embedded GPU and Multicore Processors for Emotional-based Mobile Robotic Agents

Francisco Almenar, Carlos Domínguez, Juan-Miguel Martínez,
Houcine Hassan, Pedro López

Universitat Politècnica de València
Dept. of Computer Engineering (DISCA)
Camino de Vera, 14
46071 Valencia (SPAIN)

Abstract

Control architectures based on emotions are becoming promising solutions for the implementation of future robotic systems. The basic controllers of this architecture are the emotional processes that decide which behaviors the robot must activate to fulfill the objectives. The number of emotional processes increases (hundreds of millions/s) with the complexity level of the application, limiting the processing capacity of the main processor to solve the complex problems. Fortunately, the potential parallelism of emotional processes permits their execution in parallel, hence enabling the power computing to tackle the complex dynamic problems. In this paper, Graphic Processing Unit (GPU), multicore processors and single instruction multiple data (SIMD) instructions are used to provide parallelism for the emotional processes. Different GPUs, multicore processors and SIMD instruction sets are compared to analyze their suitability to cope with robotic applications. The applications are set-up taking into account different environmental conditions, robot dynamics and emotional states. Experimental results show that, despite the fact that GPUs have a bottleneck in the data transmission between the host and the device, medium and high performance GPUs permit undertaking complex robotic problems, while low performance GPUs allows solving medium and low size problems. In addition, although SIMD instructions alone are not enough to undertake complex and some medium robotic problems, they allow obtaining some speed-up at zero

cost, just by using processor intrinsic instructions. Dual-core processors show a similar performance to SIMD instructions, while the use of quad-core processors provide similar results as low performance GPUs.
Keywords: Multicore processors, Graphics Processing Unit processors, SIMD instructions, Emotional architecture, Robotics

---

1. Introduction

Many research works [1, 2, 3, 4, 5] predict a growth of the number of intelligent robots in the industry and in our lives in the two next decades. They state that advanced robots capable of making decisions on their own as humans do are still under development and the first prototypes will not start to appear until 2030. Some researches [2, 6] state that we are seeing the emergence of the first generation robots such as the demining robot Warrior manufactured by iRobot [7], which are able to solve simple tasks with little ability to adapt to the changing environment, and running their program code on a single-core processor. However, more intelligent features that robots could include such as decision-making are not yet developed in real robots. It is expected that, by 2050, robots will be implemented using advanced computers capable of running hundreds of billions of instructions (i.e., 4th. generation robots). These robots would rival human intelligence and would be able to perform operations of abstraction and generalization, medical diagnostics, planning and decision-making [3, 4, 5].

Control architectures based on emotions are inspired on emotional natural agents. They are becoming promising solutions for the implementation of advanced robotic systems [3, 8, 9, 10, 11] because they facilitate the process of decision-making [1, 12]. They use the mechanism of emotion in organizing the behaviors, which has the following advantages: allow the robot to be autonomous to focus its attention on the most promising behavior; provide a bounded response time, which helps organizing the deliberative processes; sort the problems based on the expectations of success; autonomously adapt the

computational load to the available processor capacity allowing solving problems of increasingly complexity; separate the decision from the action processes; and use of subjective appraisal of the situation permit finding always an alternative solution. In this paper, an emotional robotic architecture for the control of complex mobile robot applications is used. In this model, two main types of processes coexist: behavior and emotion processes. The former solve the application problems (e. g., surveillance) while the latter use an emotional mechanism to motivate the robot behaviors.

Originally, all the processes of the emotional architecture, including the behaviors and the emotions were executed on a single-core computer (e. g., Intel 2,6 GHz). The emotional processes must be applied to all problems/subproblems of the robot agenda at every cycle of attention (e. g., 0.1 s). As the agenda grows in high complexity level applications, the emotional workload increases significantly as well (e.g., 200 million operations per second (MOPS)). Each one of these operations is a reduction function involving: an hyperbolic tangent function, a multiplication and a sum of up to 6 other functions. However, the control computer did not support this intensive workload because it could only execute up to 25 MOPS. Moreover, the implementation of the emotion processes on an MCU or low to medium performance DSP was discarded because these devices provided even less power computation (i. e., between 10 and 20 MOPS). Alternatively, we can use FPGAs to provide the processing capacity problem (i.e., Statrix IV by Altera). In our preliminary experiments, Statrix IV [13] was able to solve even complex problems. However, they have a high cost [14].

Fortunately, there are some alternatives. Taking into account the inherent parallelism of the problem, this paper proposes the use of multicore processors, GPUs and SIMD instructions to implement the emotional system. All these alternatives do not need of any special hardware except from the ones that can be found in any modern computer. By executing the emotion processes with the proposed alternatives, the control computer will get slack time to solve more complex applications by: (i) improving system throughput by simultaneously

executing several problems, or (ii) tackling more time critical dynamic problems (e. g., solve the problems at a higher speed).

The rest of the paper is organized as follows. Section 2 shows some related work. Section 3 describes the problem to be solved and the sequential algorithm that implements it. Section 4 describes the alternative parallel implementations explored in this paper. Section 5 present some evaluation results for different scenarios. Finally, some conclusions are drawn.

2. Related work

Control architectures inspired by the cognitive mechanism of the human mind are becoming promising solutions for developing advanced robots. One type of the cognitive architectures is based on emotions [1, 10, 11]. Different research groups [6, 12, 15] are focusing on the design of the control architectures for emotional-based robots. Salichs [12] proposes a decision making system based on drives and motivations, also based on emotions and auto-learning. The aim of the agent is to learn how to behave through the interaction with the environment, using reinforcement learning, to maximize their well being. Moshkina et al. [6] develop an algorithm based on the emotional disappointment of the robot. To achieve it, they get inspiration from the disappointment observed in animal and humans. Simulations show that robots which include this emotion are more effective than the traditional ones. Damiano [15] suggests a model where the decision making is based on a motivational system. Motivations have a value that depends on the necessity that has to be satisfied and incentives stimulates. Once all the values are calculated, the biggest motivation activates and organizes the behavior trying to satisfy the most urgent necessity. Lee-Johnson et al. [8] develop a hybrid architecture reactive/deliberative that incorporates artificial emotions to improve the decision making and the actions of a mobile robot. These emotions are active on different levels of the architecture, they modulate decisions and actions of the robot. Moshkina proposes an effective model called TAME [6] to help with the interaction between the man and the

machine. It is based on different concepts like the emotional state, the emotion and the attitude. These works propose interesting models based on emotions however, implementations of these models are usually done with sequential algorithms using general purpose processors, and consequently increasing the cost. The aim of this paper is to parallelize these emotional processes to improve performance at a reasonable cost. High- and low-performance GPUs, multicores and SIMD instructions are used to parallelize these processes.

There are some works related to the implementation of emotional systems using high performance hardware. In [16], authors propose an implementation of an emotion bio-inspired system. In this work, the authors design a FPGA controller based on emotional learning. However, the application consists of the control of a simple crane, which could be solved using a traditional PID controller. In [14], different possibilities to parallelize a limited subset of motivational processes and its implementation using a Statrix IV FPGA are proposed. The results obtained improve the implementation of the system in a single-core processor. However, the cost of migrating all the operating processes of the robot to the FPGA resulted in a quite prohibitive solution. Ducrot et al. [17] present a map estimating process with 2 depths and a partial implementation using GPU processors. They use a configuration of the Cuda-Core 448 architecture combined with dual-core processors. Their purpose is constrained to just static objects. In [18] authors presented an implementation of the R* search algorithm applied to complex planning problems, and fulfilled to reducing the cost of implementation. They propose to apply this solution to a real robot in future works. However, works where GPUs, Multicores and SIMD instructions are applied to speed up the processes that implement emotional robotic models to reduce the cost of the implementation of the 4^{th} generation robots, like this paper proposes, were not found in the bibliography.

3. Emotional Architecture

3.1. Emotional processes specification

An emotion is the process of appraising an observed situation and motivating a robot behavior to undertake this situation. Figure 1 details the emotional motivation process: (i) the emotional activation, and (ii) the emotional response. The emotional activation sets an emotional state and the emotional response builds and motivates a behavior.

(i) During the emotional activation, the observed situations (1), represented as real properties, are subjectively appraised. The appraisal process depends on the robot character. The character dynamically adjusts the parameters of this process. To calculate the appraisal of the situation (5) the robot ponders and adds (4) a set of appraisal contributions (3), which are evaluated using contribution functions (2). Equation 1 represents the i^{th} situation appraisal.

$$a_i = \sum_{k=1}^{l} w_{ak} * f_{ak}(p_k) \tag{1}$$

Where: p_k is the k^{th} property of the situation,f_{ak} is the k^{th} contribution function, w_{ak} is the weight of the function and l is the number of appraisal contribution in range of 1 to 6. The situation appraisals contribute to stablish an emotional state (9). The emotional contributions (7), evaluated with contribution functions (6), are pondered and added (8) to finally give the emotional

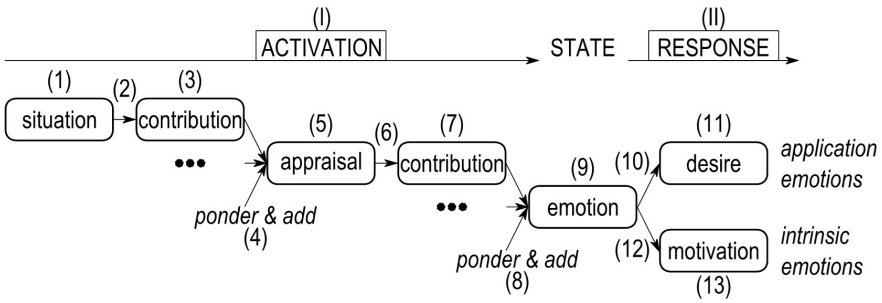

Figure 1: Emotional motivation process.

state. The emotional contributions functions are defined in the real range [-1,+1] and the emotional state in the [0, +1] range. Every emotional state is labeled in the robot navigation problem (e.g., fear of collision", a 0 level would mean no fear", while a 1 level would be afraid").

(ii) As an emotional response, the robot generates new desires (11), and motivates the behavior to accomplish the desire (13). The desires are the result of application emotions, and their response functions (10) depend on the problem, meanwhile the motivations are the result of intrinsic emotions, and their response functions (12) depend of the character of the robot.

The j^{th} emotion is expressed as shown in Equation 2.

$$s_j = \sum_{i=1}^{l} w_{ci} * f_{ci}(a_i) \tag{2}$$

Where: s_j is the state of the j^{th} emotion, f_{ci} is the i^{th} contribution function, w_{ci} is the i^{th} weight of the function.

The emotion contribution functions, f_{ci}, must have some properties such as slight variations at the ends of the range that tends to asymptotic values and abrupt variations around an inflection point in the center of the range. These properties are found in the hyperbolic tangent functions, which are used to represent the contribution functions:

$$f_{ci}(x) = th(x) = \frac{exp^{2x} - 1}{exp^{2x} + 1} \tag{3}$$

Where x is the appraisal value a_i when calculating the emotion. To allow adjusting the hyperbolic function, equation 3 is transformed in the following function

$$th^*(x) = (\frac{exp^{2(x-x_0)\delta_y} - 1}{exp^{2(x-x_0)\delta_y} + 1} - y_0) * \delta_y \tag{4}$$

where the parameters x_0, y_0, δ_x and $\delta_y$permit to translate and scale the contribution function.

These emotions are grouped in the emotional system and have the structure shown in Figure 2. The emotional system gets, at a given instant, inputs from a

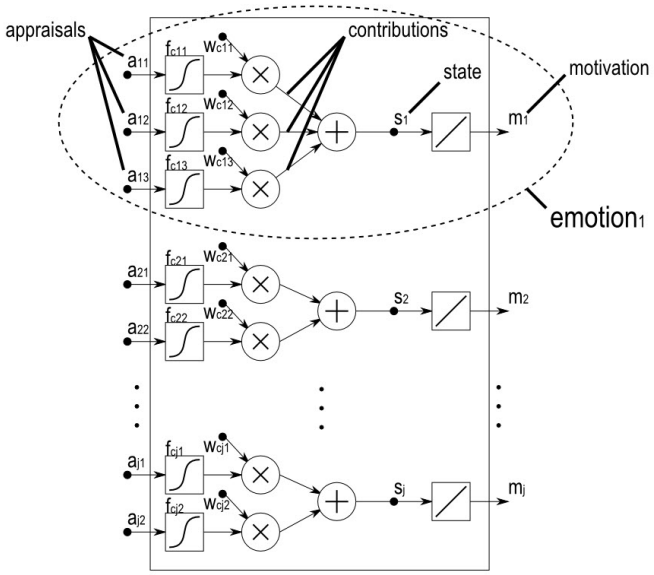

Figure 2: Emotional system structure.

set of n situation appraisals (e.g., 2M, M refers to millions) and produces a set of K motivations (e.g., 0.5M). The hyperbolic tangent is applied to each appraisal and its result is multiplied by a weight. Each emotion can be composed of up to 6 different contribution functions. The obtained emotion can pass through a final function (f_i). In this paper, though, the identity function is used (i.e. no post-processing is done). The total number of these situation appraisals in the emotional system depends on the complexity of the problem, the environment conditions and the robot dynamics. In the experiments of the multi-objective robotic applications, this number reaches a value of about 200 MOPS.

These operations can be computed in parallel since they are independent. However, in an initial implementation they were executed sequentially in a host computer controlling the robot. Due to their highly computational requirements, the capacity of the host processor was exceeded, being unable to fulfill the robot objectives.

In the next section, this paper exploits the inherent parallelism of the problem, proposing a parallel implementation that takes advantage of the hardware

already available in modern computers to implement the emotional system.

3.2. Sequential Algorithm

In this paper, the hyperbolic tangent is used to implement the emotional system of the robot (as shown in section 2.1). The following fragment of code shows an implementation in the C programming language of the calculation of the hyperbolic tangent. For the sake of simplicity, the translation and scaling factors and the weight of each contribution are not shown. This sequential code will be the basis for the parallel code that will be explained later.

```
for(i=0;i<n;i++){
    fci[i] = (exp(2*a[i])-1)/(exp(2*a[i])+1);
}


for (i=0;i<n/6;i++){
    for(j=0; j<6; j++){
        acum += fci[6*i+j];
    }
    m[i]= acum;
}
```

In the following section, different approaches to exploit the exhibited parallelism will be shown.

4. Parallel Implementation of the Emotional Architecture

4.1. Multicore Processors

The first parallel implementation of the emotional architecture is carried out using multicore processors. Multicore processors with several number of cores are standard in today's computers. The availability of several cores allows to execute in parallel several threads. To generate the parallel threads, Open MP (OMP) is used. OMP is an API that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran. This API modifies the

run-time behavior to obtain thread-level parallelism. OMP relies on directives written by the programmer that tells the compiler what can be executed in parallel. Parallelism is obtained by forking a master thread into a specified number of slave threads; these slave threads receive a part of the task that the master thread has to perform, allowing threads to run concurrently. To indicate that a loop can be executed in parallel, the preprocessor directive *#pragma omp parallel for* is written just before the for loop. The code inside the loop does not need any special modifications. In this case, the counter of the inner loop (j) and the auxiliary reduction variable ($acum$) are declared as private. The code below shows the OMP parallel version of the emotional system:

```
#include <omp.h>
...
#pragma omp parallel for private(j, acum)
for(i=0;i<n/6;i++){
    acum = 0;
    for(j=0; j<6; j++){
        fci[6*i+j] = (exp(2*a[6*i+j])-1)/(exp(2*a[6*i+j])+1);
        acum += fci[6*i+j];
    }
    m[i] = acum;
}
...
```

4.2. Graphics Processing Unit

The second parallel implementation of the emotional architecture is based on the use of a GPU coprocessor. A GPU is a parallel, multithreaded many-core processor with a tremendous computational capability. This processing power is exploited by programmers by means of a programming model. Cuda is the programming model provided by Nvidia, which is used in this paper. In this model, several blocks, each one composed of several threads, are launched to be executed in the streaming multiprocessor (SMs) available on the device

[19]. The threads of a block execute concurrently on one SM In this paper, the workload of computing the n hyperbolic tangents is split among several blocks of threads. Before the GPU starts processing, input data should be allocated in the device where it will be processed. In addition, once the processing has finished, the resulting data has to be transferred back to the host computer main memory. Both data and results are transferred between the host and the GPU and vice-versa through the PCI express bus.

The time used to perform these transfers is added to the total execution time and its impact could be important. This is the case of the emotional architecture considered in this paper, where input data is only used once per computation. In this paper, it is assumed that every time that the robot needs to calculate its emotions to make a decision, the device memory has to receive all the input data, including the one which did not changed. So, the obtained results could be considered as pessimistic. One way to improve the results is to transfer only those data that have changed since the last run, therefore avoiding useless transfers. Another way of improvement is to overlap communications and computations [20]. Anyway, even without these improvements, the obtained performance of the GPU-based implementation outstands the rest of proposals.

The code below shows the Cuda C implementation of the emotional architecture:

```
#include <cuda.h>
__global__ void hyperbolicTangent(float *dev_a, int *dev_n,
                  float *dev_fci) {

    int tid = threadIdx.x + blockIdx.x * blockDim.x;

    while(tid < *dev_n){
        dev_fci[tid] = tanh(dev_fci[tid]);
        tid += blockDim.x * gridDim.x;
    }
}
```

```
__global__ void reduction(float *dev_fci, float *dev_m,
                  int *dev_n){
    int tid = 6*threadIdx.x + blockIdx.x * blockDim.x;
    int tidemotion = threadIdx.x + blockIdx.x * blockDim.x;
    int count;

    //Each thread adds the values of 6 contiguous
    //  hyperbolic tangents, to form an emotion,
    while(tid < *dev_n){
        count = 0;
        dev_m[tidemotion] = 0;

        while(count <= 5){
             dev_m[tidemotion] += dev_fci[tid+count];
             count ++;
        }
        tid += 6 * blockDim.x * gridDim.x;
        tidemotion += blockDim.x * gridDim.x;
    }
}

int main(...){
    float *a, *fci, *m;
    float *dev_a, *dev_fci, dev_m;

    ...

    //allocate memory on the CPU
    fci = (float*)malloc( n * sizeof(float) );
    a = (float*)malloc( n * sizeof(float) );
    m = (float*)malloc( n/6 * sizeof(float) );

    //allocate memory on the GPU
```

```
    cudaMalloc( (void**)&dev_a, n * sizeof(float) );
    cudaMalloc( (void**)&dev_fci, n * sizeof(float) );
    cudaMalloc( (void**)&dev_m, n/6 * sizeof(float) );
    cudaMalloc( (void**)&dev_n, sizeof(int) );

    //copy values from CPU to GPU
    cudaMemcpy( dev_a, a, n * sizeof(float), cudaMemcpyHostToDevice );
    cudaMemcpy( dev_n, &n, sizeof(int), cudaMemcpyHostToDevice );

    hyperbolicTangent<<<blocks, threads>>>(dev_a, dev_n, dev_fci);

    reduction<<<blocks, threads>>>(dev_fci, dev_m, dev_n);

    //Copy emotions back from the GPU to the CPU
    cudaMemcpy( m, dev_m, ( N/6 ) * sizeof(float), cudaMemcpyDeviceToHost );
    ...

    cudaFree( dev_a );
    cudaFree( dev_fci );
    cudaFree( dev_m );
    cudaFree( dev_n );

}
```

Before processing the data, we have to allocate it on the device's memory. To do so, *cudaMalloc((void**) &dev_a, n * sizeof(float))* is used. This function call works similar to the function *malloc* in C, and indicates that the vector *dev_a*, which has a size of n floats, is allocated on the GPU's memory. Once the memory is allocated on the device, we are able to transfer the data from the host memory. The *cudaMemcpy(dev_a, a, n * sizeof(float), cudaMemcpyHostToDevice)* function does the work, copies all the values of the vector a in *dev_a*. Notice that the last argument indicates the direction of the transfer.

After the data is stored on the device, a function call can be done in or-

der to start the execution on the device. According to Cuda C syntax, the *hyperbolicTangent* $<<< blocks, threads >>>$ *(dev_a, dev_fci)* launches the execution on the GPU that computes the hyperbolic tangent. This function uses *dev_a* (input data) and dev_fci (output data) as parameters. During the execution, the device needs to know in which part of the shared variable dev_fci data should be read and stored. To take care of this point, the integer *tid* is used. When the function ends, the results are still in the device, so a transfer of data should be done to send the results to the host. To perform this work, the *cudaMemcpy(fci, dev_fci, n*sizeof(float), cudaMemcpyDeviceToHost)* call is used. It works similarly to the function used before, but now the direction has changed. The *reduction* $<<< blocks, threads >>>$ *(dev_fci, dev_m, dev_n)* launches the aggregation of the hyperbolic tangent in groups of 6. In both cases, there is an special parameter which is declared between $<<<>>>$, which indicates the amount of blocks and threads that is going to be used (see [21] to adjust this parameter to increase the efficiency).

Once the results are in the host, the memory allocated in the device should be released, as it is done in C. *cudaFree* call does this work.

4.3. SIMD instructions

Finally, we will propose the use of the SIMD (Single-Instruction Multiple-Data) instructions that are part of the ISA of processors since the MMX instruction set extensions were introduced by INTEL in 1999. For a better comparison, we have implemented SIMD version only in one core, even though it is possible to combine SIMD and multicore parallelism capabilities to obtain better results.

SIMD instructions allow exploiting data-level parallelism. Data-level parallelism consists of performing the same operation to different data at the same time. Data must be of a uniform type and must need the same instruction behavior. SIMD basic unit is the vector, which consist of a row of individual numbers or scalars. Regular CPUs perform operations on scalars one at a time. However, SIMD instructions operate on all the scalars of a vector as a unit, performing the same operation on each scalar. For example, considering

single-precision floating-point, which occupies 32-bit. Calculations in parallel can be done if data is grouped by 128-bit vector, allowing doing four single-precision floating-point operations at the same time [22, 23]. So, the length of the individual vectors determines the number of elements of that type that can be worked with. Streaming SIMD Extensions (SSE) is an implementation of SIMD instructions that allows working with 128-bit vectors. Advanced vector extension(AVX), a more advanced implementation, allows working with 256-bit vectors(i.e., up to 8 floats can be processed in parallel). It is only available to processors which have Intel Sandy Bridge, AMD Bulldozer architecture or newer.

The use of SIMD instructions are disabled by default. Using the *gcc* compiler, we enable the generation of these instructions by adding the *-msseX* or *-mavx* flags, where X represents the SSE version number when compiling.

The code below shows the implementation of the emotional architecture using SIMD instructions:

```
#include <emmintrin.h>
#include <mmintrin.h>

...

int main(...){
    //one is a vector composed of 1.0 values, two is composed of 2.0,
    // and zero is composed of 0.0 values
    float *aux;
    __m128 div, ptrPos, ptrNeg, ptr, ptr2, ptrEm, one, two, zero;

    ...

    posix_memalign((void**)&fci, 16, n * sizeof(float));
    posix_memalign((void**)&a, 16, n * sizeof(float));
    posix_memalign((void**)&m, 16, n/6 * sizeof(float));
```

```
    ...
    //hyperbolic tangent
    for(i=0; i<n/4; ++i){
        ptr = _mm_load_ps(&a[4*i]);
        ptr = _mm_mul_ps(two, ptr);
        ptr = fmath::exp_ps(ptr);
        ptrNeg = _mm_sub_ps(ptr, one);
        ptrPos = _mm_add_ps(ptr, one);
        div = _mm_div_ps(ptrNeg, ptrPos);
        _mm_store_ps(fci+4*i, div);
        }

    //reduction
    for(i=0; i<n/6; ++i){
        // loading values
        ptr = _mm_load_ps(&fci[4*i]);
        ptr2 =_mm_load_ps(&fci[4*(i+1)]);

        ptr2 = _mm_movelh_ps(ptr2, zero);

        // horizontal add
        ptrEm = _mm_hadd_ps(ptr,ptr2);
        ptrEm =_mm_hadd_ps(ptrEm,zero);
        ptrEm =_mm_hadd_ps(ptrEm,zero);

        _mm_store_ps(&aux[0], ptrEm);

        m[i] = aux[0];
        j++;
        }
    ...
}
```

Depending on which set of SIMD instructions are being used and its version,

a different include directive must used in the code. In this case, the SSE instructions version 3.0 are used. To declare that a variable requires 128/256-bit registers, the _m128/_m256 types should be used. The allocation of memory for data is done using the *posix_memalign((void**)&a, 16, n * sizeof(float))* function call, which ensures aligned data that leads to a better behavior. The *_mm_load_ps(&app[4])* is one of the calls that are enabled by using the gcc $-msse$ option flag. It will generate an assembler instruction that loads the first four members starting from the i^{th} pointer of a and stores them in ptr. Then, the computation of the hyperbolic tangent begins. As there is not an instruction to compute an exponential in SSE instructions, a call to a function of the *fmath* library was used [24]. In each iteration of the loop, the resulting data is stored with the *_mm_store_ps* instruction. Finally, notice that the loop last n/4 iterations, due to each iteration performs 4 hyperbolic tangents in parallel.

During the reduction of the hyperbolic tangents in groups of six, the addition of the six elements is performed by using the "horizontal add" instruction. The *_mm_hadd_ps (ptr, ptr2)* adds horizontally *ptr* and *ptr*2 (i.e., it adds the values of its operands by pairs). The three horizontal adds allows performing the sum of up to 8 operands (six of them are used in our case). The result will be stored in the i^{th} position of m.

5. Evaluation

5.1. Robot application

The emotional processor is designed to tackle mobile robotic applications. The multipurpose mobile robot performs activities such as diagnosis, transportation, cleaning, and surveillance, simultaneously.

To define the emotional computational workload of the applications, a simulator of the robot environment is used (see Figure 3). The simulator generates a large stock of scenarios to test the robotic platform while performing its activities. As an example, Figure 3 shows the result of an accident and the mobile robot trying to fix it. After the accident, there are multiple parts of a broken ob-

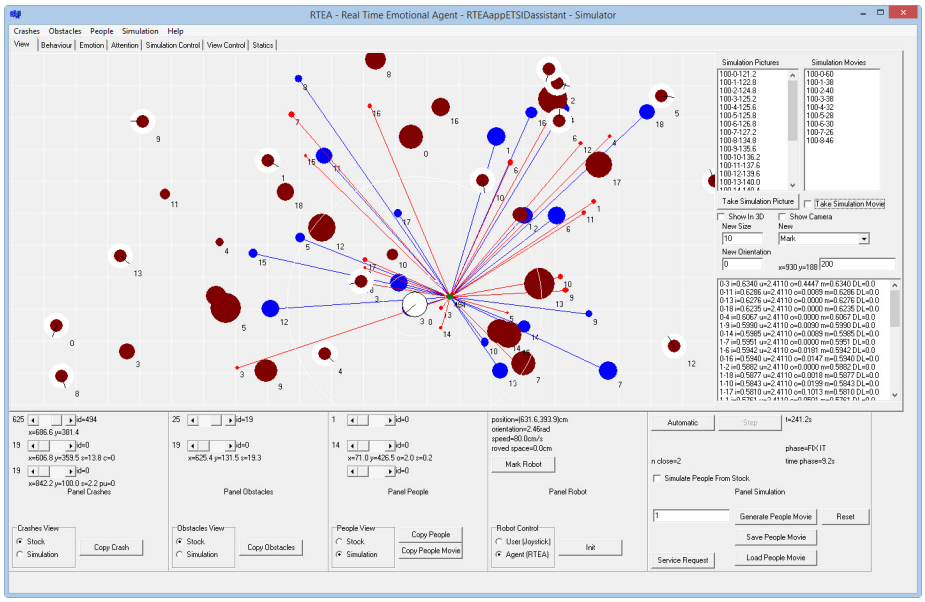

Figure 3: Robot solving a crash in the operating area.

ject and dust spots spread over the spatial area represented in the figure. To fix the accident (1) the robot (2) defines a set of sub-problems (3). It must pick the parts up and clean the spots. The emotional system of the robot motivates every sub-problem considering several appraisals about the sub-problem situation: the importance, the probability of success, the urgency, and the opportunity. The attention system of the robot uses these calculated motivation values to apply its attention policy. A set of people populates the accident scenario too (4). The simulator defines the behavior of these people as they move around the crash point. These people and other obstacles interfere the activities the robot performs. To guarantee the safety requirements and avoid collisions, the robot must perform the repairing activities adjusting its speed as the conditions of the environment change.

The robot speed, the number of objects in the operation area, and the collision risk factors define the attention cycle of the robot, T_a, which lies in the range of [0.1s , 0.5s].

Table 1 shows the robot speed values used in the experiments.

Table 1: Robot Speed.

Safe	Normal	Risky
0.1m/s	1m/s	2m/s

Table 2: Complexity of Application Problems

Simple	Normal	Complex
0.5 Mopc	1 Mopc	2 Mopc

Table 2 shows the assumed values of the complexity for three types of applications, measured in required millions of emotion operations per attention cycle (Mopc). To obtain these values, applications of different complexities are run in a simulator environment and the number of emotions involved in each of the applications is calculated. A simple problem requires the execution of about 0.5 Mopc, while a complex application involves the execution of 2 Mopc.

The emotional state of the robot represents the ratio between the time spent to execute the emotional processes and the attention cycle (T_a). Three robot emotional states are considered in the experiments (i. e., relaxed, normal and stressed). In the relaxed mode, the robot dedicates less time to the emotional processing and more time to solve application problems, whereas in the stressed mode it is the contrary. In the relaxed mode, the time used to compute emotions is less than 10% of T_a. In the normal mode, it is assumed between 10% and 25%, and in the stressed mode it is between 25% and 40%, as shown in Table 3. A workload higher than 40% will not be acceptable because the applications processes are stalled and the robot cannot fulfill the objectives.

By combining the different problem complexities, robot speeds and emotional states, the computing power of the emotional architecture can be estimated, measured in MOPS (millions operations per second, see Section 3.1). Table 4 shows these requirements.

Table 3: Robot Global Emotional State

Relaxed	Normal	Stressed
< 0.1	[0.1, 0.25]	[0.25, 0.4]

Table 4: Required Emotional Processing Power (MOPS)

		Robot speed (m/s)		
Problem	Robot state	0.1	1	2
Simple	Stressed	3	5	11
Simple	Normal	4	8	17
Simple	Relaxed	13	25	51
Normal	Stressed	6	11	19
Normal	Normal	8	17	33
Normal	Relaxed	26	49	99
Complex	Stressed	9	21	39
Complex	Normal	17	33	68
Complex	Relaxed	51	99	200

5.2. Evaluation Framework

The parallel implementation of the emotional architecture proposed in this paper has been evaluated on different platforms. The version based on the use of multicore processors has been run on a Intel core i7 processor with 4 cores running at 2.93 GHz (3.6 GHz in turbo mode) [25]. For SIMD instructions, both versions using SSE and AVX instructions were evaluated. For GPUs, two experiments were run on two different graphic cards, an Nvidia GTX 9800 and a Nvidia GTX 670. Table 5 shows the characteristics of these GPUs. For comparison purposes, results for the sequential algorithm running on one core and for an implementation based on FPGAs (e.g., statrix IV) are also shown.

The emotional based robot executing different applications, under the dif-

Table 5: GPUs characteristics

	GPU models	
characteristic	GTX 9800	GTX 670
Cuda cores	128	1344
Processor Frequency (MHz)	675	980
Memory Bandwidth (GB/sec)	70.4	192.2

ferent environmental conditions, robot emotional state and dynamics, is evaluated. The evaluation is focused in the analysis of the performance, measured on MOPS.

5.3. Evaluation results

Figures 4 to 12 show the results of evaluating the different implementation alternatives. In each Figure, the bars represent the maximum computation capacity in MOPS that each processor, SIMD, GPU or FPGA can achieve, respectively. For each pair (complexity problem, robot emotional state), the robot speed imposes a minimum computational capacity required to solve a specific problem. This is shown as the horizontal lines in the figures. For instance, in the case of a simple problem and a relaxed robot (Figure 6), if the speed is 2m/s, the minimum computation capacity required by the processors is 51 MOPS, while at 0.1m/s the required capacity is 13 MOPS (these bounds are the ones shown in Table 4). In general, for the same type of problems, at higher speeds, the computational requirements of the processors increase. On the other hand, as the complexity of the problem increases, the processor computation requirements to solve the problem also increase. Moreover, for the same kind of problems, if the emotional robot state is becoming more stressed, then the computational requirements decrease because the time dedicated to the emotional computation is higher.

For a simple problem, and a stressed robot (Figure 4), any implementation is able to fulfill the requirements in excess. This is also the case of a normal robot

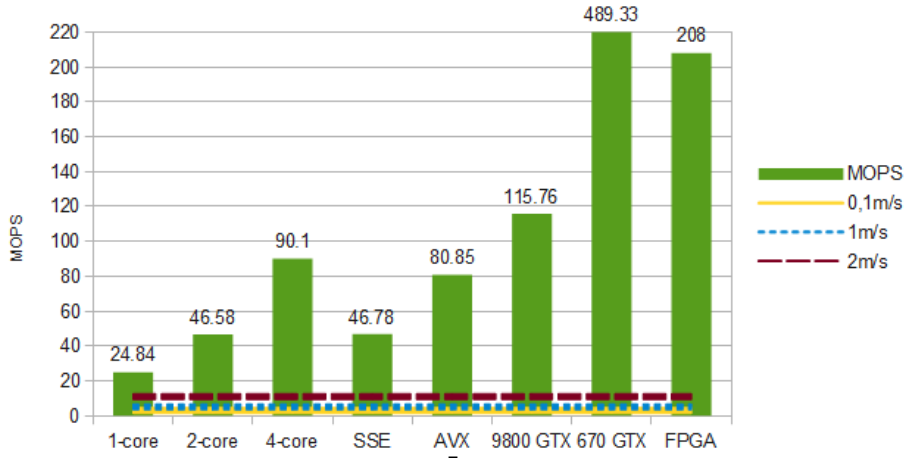

Figure 4: Simple problem. Stressed robot.

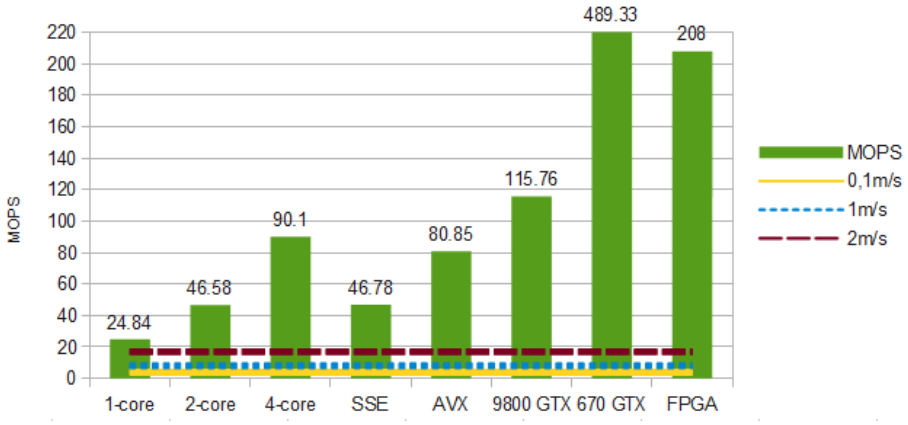

Figure 5: Simple problem. Normal robot.

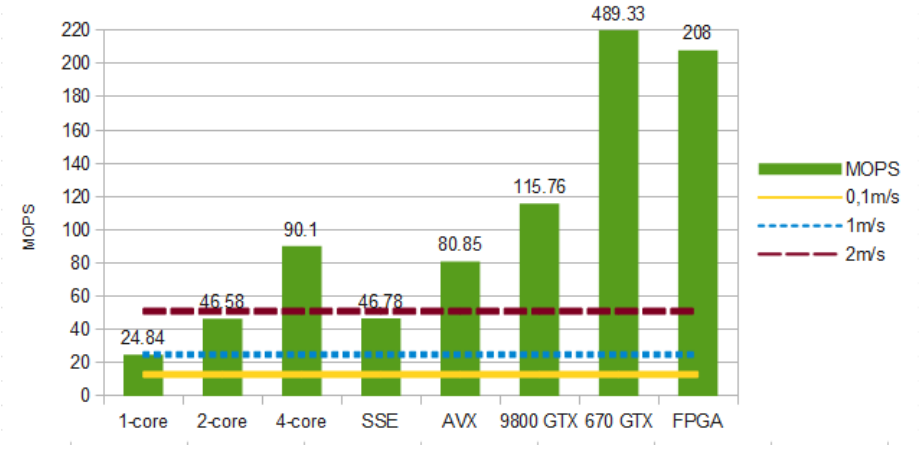

Figure 6: Simple problem. Relaxed robot.

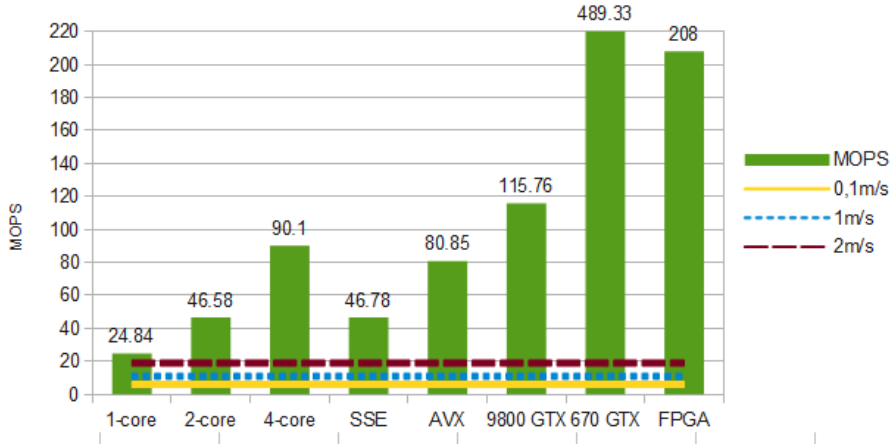

Figure 7: Normal problem. Stressed robot.

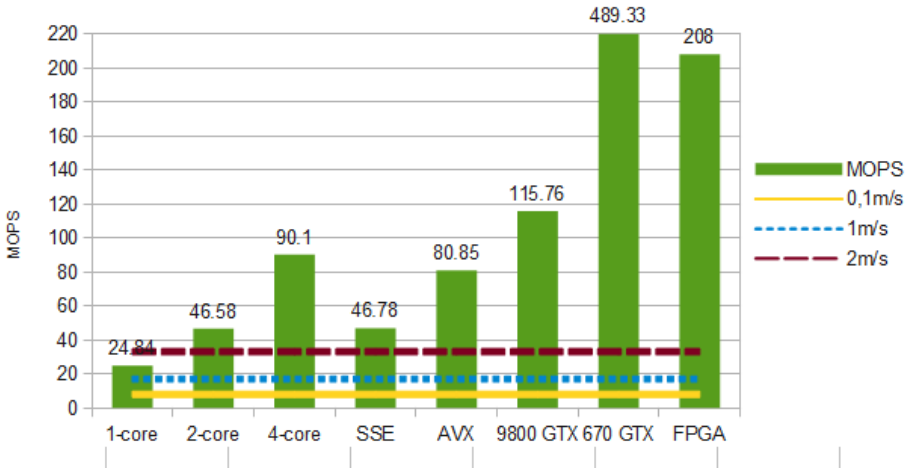

Figure 8: Normal problem. Normal robot.

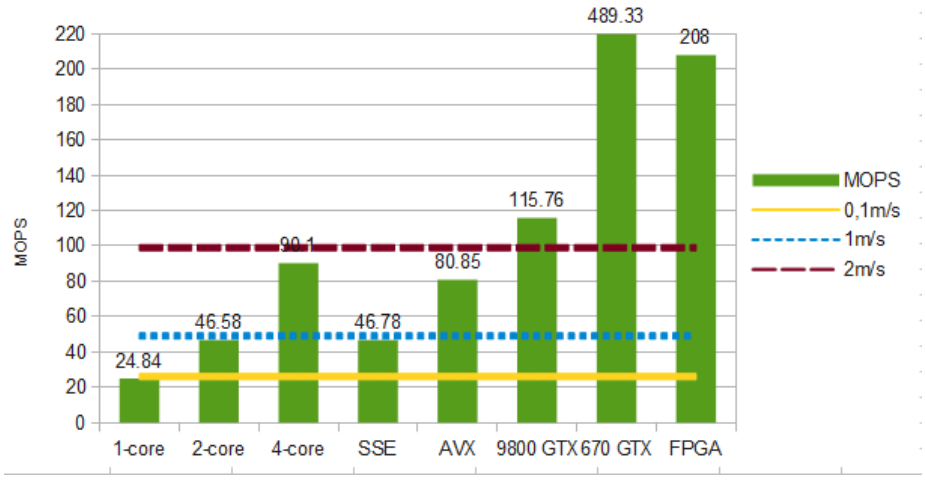

Figure 9: Normal problem. Relaxed robot.

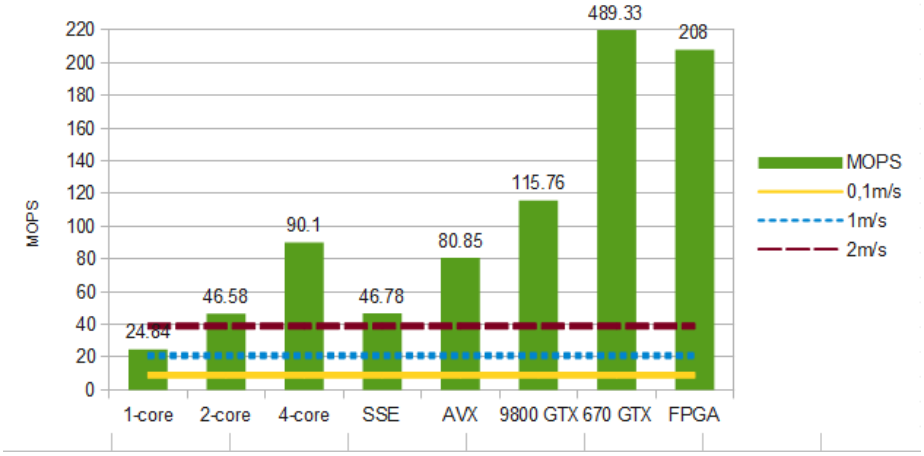

Figure 10: Complex problem. Stressed robot.

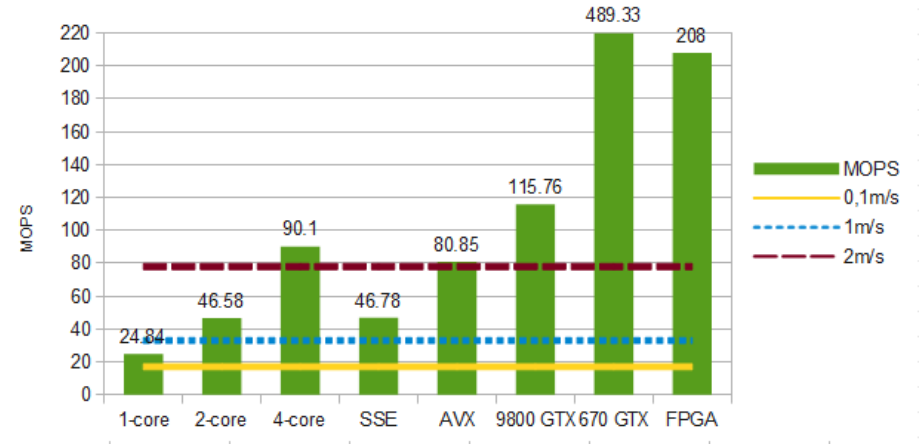

Figure 11: Complex problem. Normal robot.

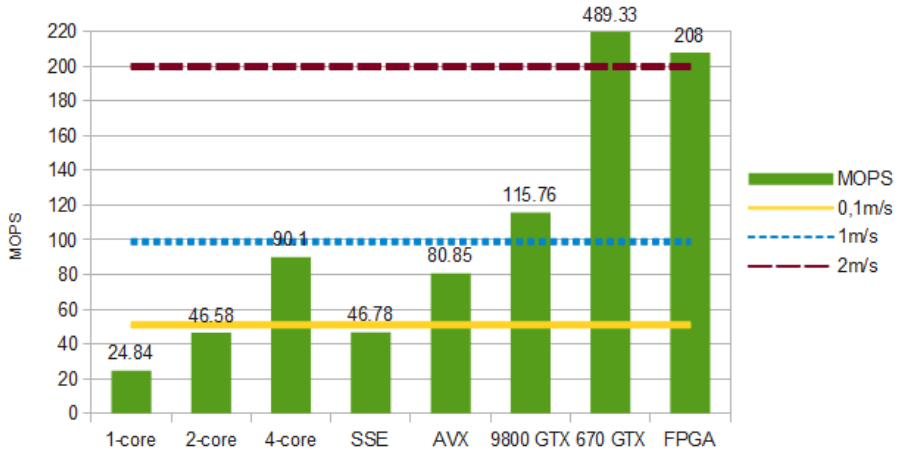

Figure 12: Complex problem. Relaxed robot.

(Figure 5). For a relaxed robot solving simple problems (Figure 6) the sequential implementation of the emotional architecture only works for the low and medium speeds. For the highest speed, one-core implementation only achieves 24.84 of the 51 required MOPS.

The sequential implementation also works for a normal problem and a stressed robot (Figure 7). However, it is unable to support a normal problem on a normal robot (Figure 8) at high speed. Any of the parallel implementations are enough in this scenario. For a normal problem in a relaxed robot (Figure 9), the results are quite different. Only GPU- and FPGA-based implementations can support the robot running at the highest speed. AVX-based and Quad-core implementations also support 1 m/s speed. Two-core and SSE-based implementation supports only 0.1m/s.

For complex problems, the sequential implementation fulfills with a stressed robot running at 1 m/s. Any parallel implementation is a good choice for a stressed robot (Figure 10). For a normal robot (Figure 11), any parallel implementation works up to 1 m/s. For the highest speed, only quad-core, AVX, GPU and FPGA based implementations work. For a relaxed robot (Figure 12) moving at 2 m/s (the most constrained requirements) only the FPGA and one

of the GPUs (GTX 670) are able to provide the required computational power.

As shown, only Nvidia GTX 670 and FPGA Statrix IV can tackle complex problems under the most constrained requirements: relaxed robot at maximum speed. However, in this case 670 GTX is a better election due to it outperforms Statrix IV in computational capabilities and its cost is much lower than Statrix IV. Nvidia GTX 9800 and Quad-core, which achieve approximately the same processing power, are the next more powerful solutions, hence they are a suitable election to solve less constrained problems than the previous one but still complex ones (e. g., complex problem and normal robot). SSE instructions are not able to tackle problems when the robot is relaxed and the robot maximum speed is required. Its performance is almost the same as a two-core processor. AVX instructions provide almost the same performance as a quad-core processor and can tackle almost the same problems as the Nvidia GTX 9800. It must be noticed that SSE and AVX instructions allows increasing the computer computational capabilities at zero hardware cost.

6. Conclusions

Emotional architectures are being considered promising solutions to implement robots of the future. However these architectures have very high computational requirements, which consumes the computational power of the main robot controller. To reduce this consumption and allow the main controller solving more complex tasks, the parallelism of the emotional processes of the architecture have been exploited and their implementation on GPUs, multicore processors and using SIMD instructions have been tackled. A mobile robotic application -under different environmental, dynamic and emotional robot state conditions, implementing an emotional-based GPU architecture has been proposed. The robotic application performances have been evaluated for Nvidia GTX 9800 and Nvidia 670, and the results are compared with a quad-Core processor (i.e., Intel i7 CPU 870 2.93GHz), SIMD instructions (i.e., SSE and AVX) and FPGA (Statrix IV). Results show that Nvidia GTX 670 and Statrix IV solve

most complex problems under the most constrained requirements, but Statrix IV is much more expensive than GTX 670. Nvidia GTX 9800 and quad-core processors solve medium size problems, while AVX instructions obtains similar performance but without any additional hardware cost; however it requires a processor with Sandy Bridge architecture or newer. SSE instructions provides roughly the same performance as a dual-core and allows tackling some of the normal problems without any additional cost.

References

[1] M. Malfaz and M.A. Salichs (2010), Using MUDs as an experimental platform for testing a decision making system for self-motivated autonomous agents. Artificial Intelligence and Simulation of Behaviour Journal, Vol. 2, No. 1, pp.21-44.

[2] L. Damiano and L. Cañamero (2010), Constructing Emotions. Epistemological groundings and applications in robotics for a synthetic approach to emotions. AI-Inspired Biology Symposium, , Leicester, UK.

[3] N. Hawes, J. Wyatt and A. Sloman (2009), Exploring design space for an integrated intelligent system. Knowledge-Based Systems, Vol. 2, No. 7, pp. 509-515.

[4] A. Sloman (2009), Some Requirements for Human-Like Robots: Why the Recent Over-Emphasis on Embodiment Has Held Up Progress. Creating Brain-Like Intelligence, 248-277, DOI: 10.1007/978-3-642-00616-6_12.

[5] H. Moravec (2009), Rise of the Robots–The Future of Artificial Intelligence. Scientific American. http://www.scientificamerican.com/article/rise-of-the-robots/. Accessed 9 Jan 2015.

[6] L. Moshkina, R. C. Arkin (2009), Beyond Humanoid Emotions: Incorporating Traits, Attitudes and Moods. IEEE International Conference on Robotics and Automation 2009

[7] iRobot industrial robots website: http://www.irobot.com/gi/ground/. Accessed 9 Jan 2015.

[8] C. P. Lee-Johnson and Dale A. Carnegie (2010), Mobile Robot Navigation Modulated by Artificial Emotions. IEEE Transactions on Systems, Man, and Cybernetics PART B: CYBERNETICS, VOL. 40, NO. 2.

[9] C. Domínguez, H. Hassan, A. Crespo (2009), Real-time Emotional Agent Architecture Application on Service Mobile Robot Control The 2009 International Conference on Artificial Intelligence. Las Vegas, Nevada, USA.

[10] E. Daglarli, H. Temeltas, M. Yesilogly (2009), Behavioral task processing for cognitive robots using artificial emotions Neurocomputing 72(13):28352844.

[11] R. Ventura, C. Pinto-Ferreira (2009), Responding efficiently to relevant stimuli using an emotion-based agent architecture Neurocomputing 72(13):29232930.

[12] M.A. Salichs, M. Malfaz (2012), A new approach to modelling emotions and their use on a decision making system for artificial agent. IEEE Transactions on affective computing, 3(1):5668.

[13] Altera Corporation (2014), Stratix IV device handbook, vol 14, version 5.9. http://www.altera.com/ literature/lit-stratix-iv.jsp. Accessed 14 Oct 2014.

[14] C. Dominguez, H. Hassan, A. Crespo and J. Albaladejo (2014), Multicore and FPGA implementations of emotional-based agent architectures, The Journal of Supercomputing, DOI: 10.1007/s11227-014-1307-6 (published Online October 2014).

[15] L. Damiano and L. Cañamero (2010), Constructing Emotions. Epistemological groundings and applications in robotics for a synthetic approach to emotions, Proceedings of international symposium on aIinspired biology, The Society for the Study of Artificial Intelligence, pp 2028, Leicester, UK.

[16] M.R. Jamali , A. Arami, M. Dehyadegari, C. Lucas, Z. Navabi (2009), Emotion on FPGA: Model driven approach, Expert Systems with Applications 36 (2009) 7369?7378.

[17] A. Ducrot, Y. Dumortier, I. Herlin, V. Ducrot (2011), Real-time quasi dense two-frames depth map for Autonomous Guided Vehicles, Intelligent Vehicles Symposium (IV), 2011 IEEE, vol., no., pp.497-503, 5-9 June 2011

[18] J.T. Kider, M. Henderson, M. Likhachev, A. Safonova (2010), High-dimensional planning on the GPU, 2010 IEEE International Conference on Robotics and Automation (ICRA), pp.2515-2522, 3-7 May 2010.

[19] Cuda (2010), NVIDIA CUDA C Programming Guide, NVIDIA, Nov 2010.

[20] M. Harris (2012), Nvidia developer zone, http://devblogs.nvidia.com/parallelforall/how-overlap-data-transfers-cuda-cc/

[21] J. Luitjens, S. Rennich (2011), CUDA Warps and Occupancy, http://on-demand.gputechconf.com/gtcexpress/2011/presentations/cuda_webinars_WarpsAndOccupancy.pdf

[22] J. Stokes (2000), SIMD architectures, http://arstechnica.com/features/2000/03/simd/2/

[23] C. Lomont, Introduction to Intel Advanced Vector Extensions, http://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions

[24] M. Shigeo (2010), Fast approximate float function Fmath, http://homepage1.nifty.com/herumi/soft/fmath.html

[25] Intel Core i7-870 Processor Specifications, http://ark.intel.com/products/41315/IntelCorei7870Processor8MCache2_93-GHz