A Procedural Long Term Memory for Cognitive Robotics

Optimizing Adaptive Learning in Dynamic Environments

R. Salgado, F. Bellas, P. Caamaño, B. Santos-Díez, R. J. Duro
Integrated Group for Engineering Research, Universidade da Coruña
Ferrol (A Coruña), Spain
rodrigo.salgado@udc.es, fran@udc.es, pcsobrino@udc.es, borja.santosdiez@udc.es, richard@udc.es

***Abstract*—This paper provides some insights into the advantages of using a Long-Term Memory (LTM) for optimizing the adaptive learning capabilities of a cognitive robot in dynamic environments. Specifically, a procedural LTM that stores basic models and behaviours is included in the evolutionary-based Multilevel Darwinist Brain (MDB) cognitive architecture. The memory system is based on learning error stability and instability to detect if a model is candidate to enter the LTM or to be recovered. A LTM replacement strategy has been developed that is based on context detection using functional comparison of the models' response. The LTM elements are tested in theoretical functions and in a simulated example using the AIBO robot in a dynamic context with successful adaptive learning results.**

Keywords-Adaptive Learning, Cognitive Robotics, Evolutionary Computation, Dynamic Environments

I. INTRODUCTION

In the last decade, several authors in the autonomous robotics field have started to introduce in their robots biological brain models taken from neurophysiology trying to computationally reproduce the complex processes occurring within the brain with the aim of obtaining robots with a higher level of autonomy [1][2][3]. This approach is different from the traditional one of studying brain operation and implementing the resulting functionalities from an engineering perspective.

Brain-like robotic approaches are typically included under the name of *cognitive robotics*, due to the fact that such robots are characterized by their cognition. A cognitive robot is characterized by its capacity of acquiring knowledge in an autonomous and adaptive way, that is, a cognitive robot has adaptive learning capabilities and, as a consequence, its behaviour is really autonomous.

One of the most studied aspects of biological brains in cognitive robotics is memory [4][5][6], as the main cognitive functions depend on it. In planning and decision-making, memory is used as the substrate for the creation of models where predictions can be carried out. Action selection is guided by previously selected actions and their consequences. Other cognitive processes such as attention, visual processing, and reasoning are strongly coupled to the memory system. Even emotions affect memory by modulating the behaviour of the neural cells [7]. Finally, as the key aspect for the work presented here, memory plays a basic role in learning, and although it is possible to learn without explicitly using previous experience, the retention of learned patterns is clearly improved using memory and, consequently, the behaviour in dynamic contexts is optimized.

Cognitive architectures are the computational implementation of a cognitive model, and as such, constitute the substrate for all cognitive functionalities in robots, like perception, attention, action selection, learning, reasoning, etc [8]. Most traditional cognitive architectures, such as *SOAR* [9], *LIDA* [10], *Micro-PSI* [11], *OpenCogPrime* [12] or *IMA* [13], contain memory systems based on theoretical concepts from neuroscience and cognitive psychology where two main types of memory are identified, *Short-Term Memory (STM)* and *Long-Term Memory (LTM)*. STM is composed of a sensory memory that contains the information perceived by the perceptual system through the sensors and a working memory that it can be considered as a cache that actively maintains information relevant to the current task for a short period of time. Thus, STM stores the conscious information for a short period of time, in which it undergoes a process of consolidation until it is permanently stored in the LTM.

LTM preserves a large amount of information for a very long time. It is typically divided into *procedural* or *implicit* memory, related with implicit knowledge and non-conscious processes like the tuning of motor skills, and *explicit,* which stores conscious knowledge. Explicit LTM can be subdivided into *Semantic*, which stores general knowledge about the world, including facts and properties of objects, and *Episodic*, which stores facts, events or episodes that occurred during a lifetime contextualized in time and space.

LTM is the most complex memory element and it has received much attention in cognitive robotics research [14]. Regarding explicit LTM, only a few cognitive architectures consider semantic memory systems appropriately due to the complexity of transforming events or facts into knowledge [6][13], that is, the classical problem of assigning meanings to the perceptions. The case of episodic memory is different, as this LTM module the one that has been studied more in depth and implemented in real robots [9][10], although in tasks of limited complexity [4][5].

Most cognitive architectures [14] include a procedural memory element that contains the basic knowledge of how to select and perform basic actions or behaviours. But in all of

them, such basic knowledge is pre-defined and inserted in the robot memory. Moreover, it remains fixed during the robot lifetime, so new skills cannot be included in the procedural memory. In this work we are interested in this specific aspect of memory in cognitive robotics: providing an existing cognitive architecture with a long-term procedural memory that supports adaptive learning of basic models and skills throughout the robot's lifetime. In particular, we will analyse the optimization in the learning processes that can be achieved using these structure when dealing with dynamic environments.

The remainder of the paper is structured as follows: section II contains a brief introduction to the current version of the Multilevel Darwinist Brain cognitive architecture that is used in this paper as a base architecture. Section III is devoted to the detailed description and theoretical test of the procedural long-term memory that has been added to the architecture. In section IV we have included a practical example of adaptive learning in a dynamic environment with a simulated robot. Finally, section V summarizes the main contributions of this work.

II. MULTILEVEL DARWINIST BRAIN

The Multilevel Darwinist Brain (MDB) is a cognitive architecture that has been widely applied to different real robot learning problems [15]. It is based on a utilitarian cognitive model that allows a general autonomous agent to decide the actions it must apply in its environment in order to maximize its satisfaction (degree of fulfilment of the task). The MDB uses three types of functions, called world, internal and satisfaction models [15] to select the action it will execute through an internal reasoning process. The world and internal models predict, respectively, the external and internal sensorial values in instant t+1 from the sensorial values and the action applied in instant t. From these predicted sensorial information, the satisfaction model predicts the satisfaction of the robot associated to the action applied.

The world, internal and satisfaction models that are used for action selection in the MDB are not pre-defined, and they must be learned during the robot's lifetime while it is interacting with its environment. This approach is typical in the Cognitive Developmental Robotics field [2], a cognitive robotics subfield where an embodied and autonomous development of the robotic "brain" is the key aspect. The main originality of the MDB is in the use of evolutionary algorithms (EA) operating in real time for the adaptive learning of the models and of artificial neural networks (ANN) as the substrate for those models [15].

Fig **1** displays a functional diagram of the MDB in its current version, including the Long-Term Memory element that is the topic of this work and that will be explained in the next section. As we can see, the MDB is structured into two different time scales, one devoted to the execution of the actions in the environment in real time (execution scale) and the other with the learning of the models (learning scale). The elements present in the execution scale necessarily run in the robot's physical hardware while the elements present in the learning scale could run on it if the computational power allows it, but they could run in distributed computers. The operation of the MDB can be described in terms of these two scales.

Starting from the learning scale, let us assume that the robot has executed an action in the environment and a new *iteration* starts on time t. We have new perception values that are introduced in the *Short-Term Memory (STM)*. These real world samples or episodes are stored in an *episodic buffer (EB)* (one for each type of model) if they pass an *attention* mechanism that acts as a filter and replacement mechanism with the aim of storing the most relevant episodes for learning in each case. The *EB* has a very limited capacity according to the temporal nature of the *STM*.

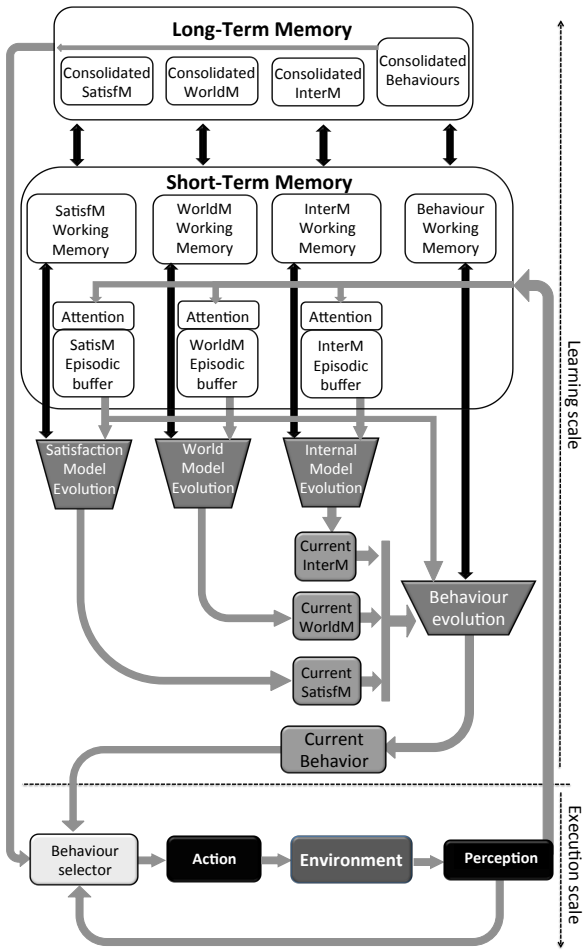

Fig 1. MDB functional diagram including the developed LTM

Once the episode is stored or discarded, the *model evolution* starts for each type of model using as population the models stored in the *Working Memory (WM)*. This evolution is carried out during a small number of generations to avoid a premature convergence of the model towards a specific *EB*. The evolved models are, as commented above, ANNs that are randomly initialized and stored on their corresponding *WM* in the first iteration and imported from the *WM* in subsequent iterations.

After the evolution finishes, the best models are selected as *current world, internal and satisfaction models* and they make up the internal representation of the robot's world in iteration *t.* These models are used in the *behaviour evolution*. A behaviour in the MDB is a function that provides the action to be applied in a given instant of time from the sensorial information

available at that point. In the first MDB version, every time the model evolution finished, an action selection process was carried out to choose the best action to be applied. Now, this process has been generalized with the inclusion of behaviours. To evolve them, the current models are used to compute the fitness function, because they provide the satisfaction associated to a given action, so the behaviours can be evaluated using all the episodes stored in the *EB*. As in the case of the models, the behaviours are represented by ANNs that are stored in the *behaviour WM* and evolved during a small number of generations. Once the behaviour evolution finishes, the best individual is selected as the *current behaviour* and transferred to the execution scale to be used to select the action.

Every time the robot executes an action new episodes from the real environment are obtained. As more iterations take place, the MDB acquires more information and thus the model learning processes should produce better models and, consequently, the behaviours obtained using these models should be more reliable, and the actions provided by them more appropriate to maximize the satisfaction.

In the execution time scale, the robot has a *behaviour selector* that chooses, as we will explain later, the behaviour that will be applied in real time from the one obtained in the learning scale or from others coming from the Long-Term Memory. In any case, once a behaviour is selected, it simply provides the action to be applied as a function of the sensorial information acquired in the previous iteration. This behaviour is used until one that produces a better predicted satisfaction for the current circumstances of the robot is obtained in the learning stage or is found in the LTM.

III. PROCEDURAL LONG TERM MEMORY

The initial version of the MDB without LTM is very inefficient in dynamic environments. This is due to the fact that the models that were learned were not stored, and, consequently, in the case of returning to a previous context, they must be relearned again. This is the main motivation behind the implementation of a Long-Term Memory for a cognitive architecture. Moreover, previously learned models can be used as seeds for new contexts that could appear as simple variations of others (for example, two environments with small differences in the floor features).

The models in the current version of the MDB correspond to cognitive processes, those of finding relations between sensorial information and satisfaction. Taken as a whole, the current models represent the knowledge about the world and itself the robot has acquired. On the other hand, the behaviours may be associated to the basic *skills* and *habits* of the robot. To improve on the efficiency of the MDB, we decided to implement a procedural long-term memory (LTM) that stores knowledge on the world, skills and habits in the form of models and behaviours.

First of all, it must be pointed out that the LTM management is independent for each type of model, which, in fact, run in different threads concurrently [16]. This has been represented in the diagram of Fig **1** trying to maintain each model elements in a common vertical line. We imposed a set of requirements for the LTM design (in what follows, the behaviours will be considered as models). First, it must store only one model for each context. Assuming that the robot will be placed in a dynamic environment, we must avoid storing models of intermediate contexts. Second, it must store models that have been clearly learned and that provide successful prediction results. Finally, it must allow the recovery of models in the case of detecting a change of context. As commented above, the models can be directly applied in previously learned contexts or serve as seeds for new learning processes.

To achieve these requirements, we have developed a LTM system that can be decomposed into three basic components:

A. Error stability test

To decide if a model must be stored in the LTM we perform an error stability test. The objective of this test is to find out if the current model of a given iteration is stable or not in terms of its convergence tendency. Considering an *EB* of capacity C_{eb}, the MDB learning process requires approximately $2C_{eb}$ iterations to improve. Consequently, the error stability tests start after $2C_{eb}$ iterations. From this moment onwards, after the evolution of each model finishes, we store the error obtained by the current model in an *error memory*, which has a capacity of C_{em}. Once this error memory is full, the test can start: the current model error E_{cm} is compared one by one with all the errors stored in the error memory E_{em}^{i}. We establish a percentage *stability threshold* T_s, so the current model is considered as stable if:

$$(1 + T_s) \cdot E_{em}^{i} > E_{cm} \text{ and } (1 - T_s) \cdot E_{em}^{i} < E_{cm}$$
$$\text{for } 1 < i < C_{em}$$

To analyse the response of this stability test with different parameters, we have designed a learning experiment with a theoretical function, in this case a Gaussian one. We simulate the model learning of the MDB by randomly extracting samples from a function that represent episodes and are stored in the *EB*. Each time a sample is obtained, a new learning iteration starts, as in the case of MDB. We use a Differential Evolution algorithm to adjust the parameters of a multilayer perceptron ANN that must model the theoretical function. Table 1 displays the specific parameters and function used in this case.

Target function	$z = 3e^{-\frac{x^2+y^2}{2}}$
EB size (C_{eb})	20
ANN size	2-4-1
WM size (C_{wm})	50
MDB iterations	2400
Generations per iteration	2

Table 1. Parameters of the stability experiment

Fig **2** represents the evolution of the mean squared error provided by the best model each iteration. We have placed a black point over the curve in those iterations where a model is detected as stable. The top graph corresponds to C_{em}=20 and T_s =0.3. As shown, the stability test is too soft and stability is detected even in zones where the convergence is clearly

unstable. The bottom graph corresponds to C_{em}=80 and T_s =0.1, a much harder situation to detect stability, and as a consequence it is only detected around iteration 750. It must be pointed out that in both cases the error level obtained was good enough to model the theoretical function.

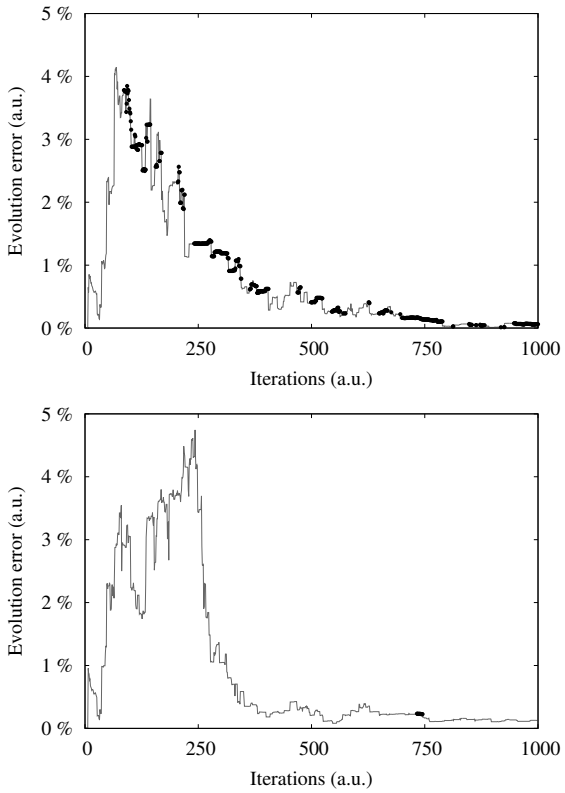

Fig 2. Evolution error for a Gaussian function with points marking the iterations where a model was detected as stable. Top graph with C_{em}=20 and T_s =0.3 and bottom graph with C_{em}=80 and T_s =0.1

Therefore, the stability test is highly sensitive to the values of error memory size and stability threshold. The ideal situation, verified in our experiments, has been an intermediate case, like the one shown in Fig **3** with C_{em}=20 and T_s =0.1.

B. Replacement strategy

If a model is detected as stable and the LTM is empty it will be directly stored. In subsequent iterations, if a model is detected as stable, and taking into account that we want to store only one model per context, a replacement strategy is required. As we are storing the genotype of an ANN, a direct comparison is not possible because genotypically different models could correspond to the same context. Consequently, we have developed a strategy that is based on a cross comparison of errors in the associated contexts. Thus, together with the ANN parameters that make up the model, we store the corresponding *EB* that represents the context where this model was stable.

Specifically, the following errors must be considered for this replacement strategy:

- E_{cm}: the current model error in the current *EB*
- E^i_{ltm}: the ith-model of LTM error in its associated *EB*
- $E^i_{cm\text{-}ltm}$: the current model error in the *EB* of the ith-model of LTM
- $E^i_{ltm\text{-}cm}$: the ith-model of LTM error in the current *EB*

When the current model of iteration *t* is detected as stable, it is marked as *acceptable* and it is executed over the *EBs* of all the models stored in the LTM. For the ith-model in the LTM the following situations may occur:

- $E_{cm}<E^i_{ltm\text{-}cm}$ and $E^i_{cm\text{-}ltm}<E^i_{ltm}$. In this case, the current model is better than the ith-model in both contexts, so the ith-model is eliminated from LTM and the current model remains as *acceptable*.
- $E_{cm}> E^i_{ltm\text{-}cm}$ and $E^i_{cm\text{-}ltm}>E^i_{ltm}$. This is the opposite case, so the current model is *unacceptable* and, consequently, it is discarded.
- $E_{cm}<E^i_{ltm\text{-}cm}$ and $E^i_{cm\text{-}ltm}>E^i_{ltm}$. In this case, each model is better in its own context, but even so they could correspond to the same situation and this must be discriminated. To do it, we have established a percentage *error similarity threshold* T_{es} so we can compare if the crossed and own errors are similar. Specifically, we can compute whether:

$$E_{cm} < (1+T_{es})E^i_{cm\text{-}ltm} \text{ and } E_{cm} > (1-T_{es})E^i_{cm\text{-}ltm}$$

or

$$E^i_{ltm} < (1+T_{es})E^i_{ltm\text{-}cm} \text{ and } E^i_{ltm} > (1-T_{es})E^i_{ltm\text{-}cm}$$

If this logical expression is true, we assume that both models correspond to the same context. In this case, we create a combined *EB* and execute both models over it, discarding the model with a worse error. If the expression is false, we assume that the models correspond to different contexts and the current model remains as *acceptable*.

This cross comparison of errors is performed for all the models in the LTM, and if the current model is *acceptable* at the end, it will be stored. As we can see, the replacement strategy includes a generalization procedure where the more general models replace those more specific.

To show a typical result of the response obtained with this replacement strategy, in Fig **3** we have represented again the evolution of the mean squared error provided by the best model in the same learning example commented above, but now using C_{em}=20 and T_s =0.1 (the other parameters are those displayed in Table 1). The iterations where stability is detected are marked over the curve too. We have included in the figure a secondary y-axis and horizontal lines as an indication of the presence of a model in LTM. For example, iteration 291 is the first where stability is detected. Consequently, a horizontal line is represented with a value of 291 in the secondary y-axis. This line is maintained until iteration 311 where another model is detected as stable and the replacement strategy decides that it corresponds to the same context and it replaces the old one. This is why from iteration 311 onwards, the horizontal line of iteration 291 disappears.

An interesting situation occurs, for example, from iteration 669 to 782 where three models are present in the LTM

(corresponding to iterations 651, 659, 668). At iteration 782, a new model is detected as stable and it generalizes the other three, which are replaced by it.

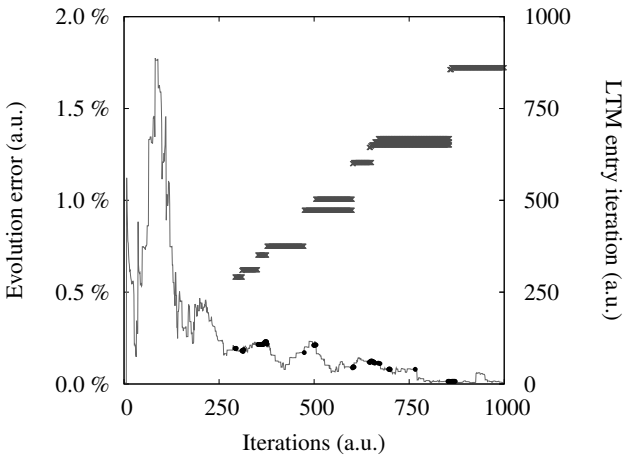

Fig 3. Evolution error for a Gaussian function with points marking the iterations where a model was detected as stable and with C_{em}=20 and T_s =0.3. The secondary y-axis corresponds to the iteration where a model is stored in LTM.

C. Model recovery

The models that are stored in the LTM are inserted in the model working memory, replacing the worse models, to be used as seeds for new learning processes. With this procedure, in the case of returning to a previously learned context, the injected model will be directly selected due to the high fitness it will obtain. In addition if the context is similar, it will take very few iterations for one of the previous models to adapt.

The key point for model recovery is deciding when it must occur. We have decided to implement an *error instability test* similar to that used in the case of selecting a model to store. In this case, we want to detect when a model learning process does not converge and the error clearly changes. We assume that this fact corresponds to a change of context, and once detected, we inject the models from LTM in the corresponding working memory.

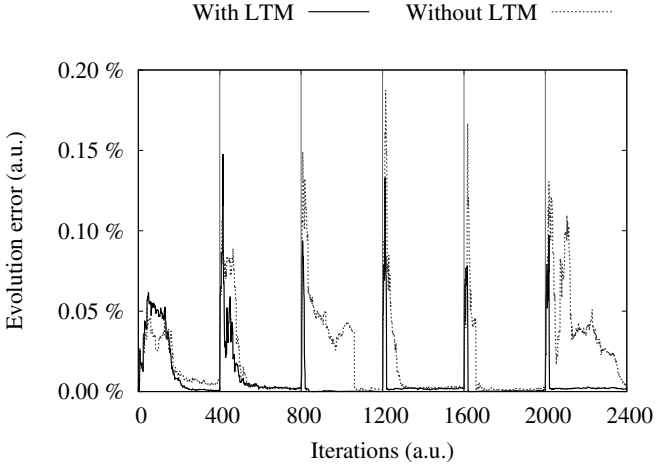

Fig 4. Evolution error for Gaussian and sinusoidal functions that are interchanged every 400 iterations

Specifically, the error instability test starts once a model is stored in LTM and it uses the same error memory as the stability one. The current model error E_{cm} is compared with highest error E^h_{em} stored in the error memory and with the standard deviation σ_{em} of the error memory. We establish a percentage *instability threshold* T_i and a number of *instability iterations* I_i, so instability is detected if:

$$E_{cm} > T_i \cdot (E^h_{em} + \sigma_{em}) \text{ during } I_i \cdot C_{eb} \text{ iterations}$$

Reliable and efficient instability detection is crucial for adaptive learning. Consequently, this test is quicker than the stability one and seeks for relevant peaks in the error convergence tendency. But to avoid detecting a punctual error as instability, we impose that the instability test must be true during a consecutive number of iterations, which is a factor Ii (instability iterations) of the *EB*.

If instability is detected, in addition to injecting all the models from the LTM into the working memory, the *EB* must be purged of episodes from the previous context to avoid learning in mixed contexts. As we can see, there is an interplay between STM and LTM that has been represented in Fig **1** with the double direction arrows that connect these two memories.

To show the general behaviour of the learning system with the LTM in dynamic contexts, we have added a second function ($z = \frac{3}{2} \cdot sin(\frac{x+y}{2}) + \frac{3}{2}$) to the learning example described in the two previous subsections (the remaining parameters are those displayed in Table 1). These two functions are changed every 400 iterations, simulating an abrupt change of context. Fig **4** displays the evolution of the mean squared error in the case of using LTM (continuous line) and without LTM (dotted line), which corresponds to the previous version of the MDB. If we consider the dotted line, it can be seen that each time a change of context occurs, the model learning process starts again. In the case of the continuous line, learning occurs only the first time the two contexts are presented, and in subsequent repetitions the models are inserted from LTM and the adaptation to the new context is immediate.

To finish this section, a very relevant consequence of the LTM implementation in the execution scale of the MDB (Fig **1**) must be highlighted. If the action selected by the behaviour in execution provides a low satisfaction value repeatedly, we can assume that a change in the context has occurred, and the *consolidated behaviours* stored in LTM can be compared to the current behaviour, in the episodes where the old behaviour failed, to select the most appropriate one. The idea behind this *behaviour selector* is that of optimizing the robot response in the case of dealing with a new context. If this context was learned before, the LTM behaviours will perform successfully quickly and if the context is new the robot will not behave properly until the learning scale obtains a new behaviour.

IV. Simulated Robot

The adaptive learning response of the MDB with the developed LTM was tested in a robotic simulated experiment. We have used the Webots simulator with an AIBO model modified by us [16] in a simple example of learning in dynamic contexts. Specifically, we place the AIBO robot in an empty scenario that only contains a pink ball (Fig **5**). Initially, the

satisfaction (S_1) is maximum when the robot reaches the ball frontally, that is, when the distance to the ball is minimum and the angle is zero (left image of Fig **5**). The change of context consists in changing the satisfaction to another one (S_2) that maximizes escaping from the ball (right image of Fig **5**).

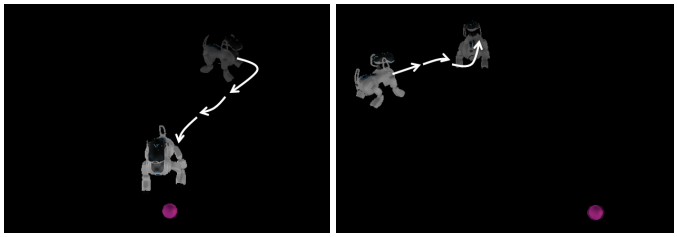

Fig 5. AIBO in simulation with functions S_1 (left) and S_2 (right)

The sensorial information considered in this case is the distance and angle to the ball (obtained from the camera and head angle), and the action is simply the angular velocity. Thus, the MDB uses two world models, one for the distance and another for the angle, and one satisfaction model. This last model has 2 inputs (distance and angle predicted by the world models) and 1 output (predicted satisfaction).

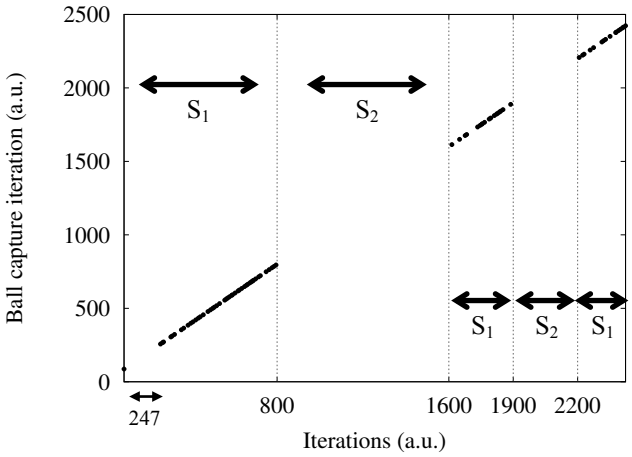

Fig 6. Iterations where the ball is captured in a changing context

The satisfaction function S_1 is maintained until iteration 800 where it is abruptly changed for S_2 until iteration 1600. At iteration 1900 S_1 is used again, then S_2 until iteration 2200, and finally, S_1 again until the final iteration 2500. Fig **6** represents the iterations where the ball was captured. It can be seen how there is an initial gap of 247 iterations in the initial phase S_1 until the ball is reached for the first time (model learning stage). This gap does not exist in the two following contexts with S_1 because the LTM system works properly, the instability is detected and the satisfaction models learned in the first phase, and stored in the LTM, are recovered providing an optimal adaptation.

V. CONCLUSIONS

A procedural Long-Term Memory (LTM) has been designed and implemented in the Multilevel Darwinist Brain (MDB) cognitive architecture. In this work we have described its main elements and operation in theoretical examples of adaptive learning and in a simulated experiment with the AIBO robot using the whole MDB. The advantage of having this type of memory element to optimize the adaptive learning capabilities of the robot and, consequently, its autonomy level has been clearly shown. We are now working on improving its operation and automating the determination of the values for the different parameters by relating them to the robot's states or even emotions.

ACKNOWLEDGMENT

This work was partially funded by the Xunta de Galicia and European Regional Development Funds through projects 09DPI012166PR and 10DPI005CT.

REFERENCES

[1] N. Kasabov, L. Benuskova, and S. Wysoski, "Biologically plausible computational neurogenetic models: Modeling the interaction between genes/proteins, neurons and neural networks," J. Comput. Theoret. Nanosci., **2** (4), 569–575, 2005.

[2] M. Asada, K. F. MacDorman, H. Ishiguro, Y. Kuniyoshi, "Cognitive developmental robotics as a new paradigm for the design of humanoid robots". Robotics and Autonomous Systems 37**,** pp. 185-193, 2001.

[3] J. L. Krichmar and G. M. Edelman, "Principles underlying the construction of brain-based devices". In Proceedings of AISB 2006*,* 2, pp. 37–42, 2006.

[4] E. C. de Castro, R. R. Gudwin, "An Episodic Memory for a Simulated Autonomous Robot", In Proceedings of Robocontrol 2010, 1-7, 2010.

[5] S. Jockel, M. Weser, D. Westhoff, J. Zhang, "Towards an Episodic Memory for Cognitive Robots", In Proceedings of the 6th International Cognitive Robotics Workshop at ECAI08, IOS Press, pp. 68-74, 2008.

[6] F. Dayoub, T. Duckett, G. Cielniak, "Toward an Object-Based semantic memory for Long-Term operation of mobile service robots", In Workshop on Semantic Mapping and Autonomous Knowledge Acquisition, IROS, Taipei, Taiwan, 2010.

[7] E. R. Kandel, J. H. Schwartz, T. M. Jessell, "Principles of Neural Science, 4th int. edition", McGraw-Hill, 2000.

[8] Bram van Heuveln, "What is Cognitive Robotics?" http://www.cogsci.rpi.edu/~heuveb, Cognitive Science Department, Rensselaer Polytechnic Institute, 2010.

[9] J. E. Laird, "Extending the Soar Cognitive Architecture", In Proceeding of the 2008 conference on Artificial General Intelligence, IOS Press, Amsterdam, The Netherlands, pp. 224-235, 2008.

[10] S. Franklin, "Cognitive robots: perceptual associative memory and learning", In IEEE International Workshop on Robot and Human Interactive Communication (ROMAN), pp. 427–433, 2005.

[11] J. Bach, "Principles of Synthetic Intelligence PSI: An Architecture of Motivated Cognition", 1st ed. New York, NY, USA: Oxford University Press, Inc., 2009.

[12] B. Goertzel, H. de Garis, "XIA-MAN: an extensible, integrative architecture for intelligent humanoid robotics", In Proceedings of the BICA-08, pp. 86–90, 2008.

[13] K. Kawamura, S. Gordon, P. Ratanaswasd, E. Erdemir, J. Hall, "Implementation of Cognitive Control for a Humanoid Robot", International Journal of Humanoid Robotics, **5** (4), pp. 547-586, 2008

[14] B. Goertzel, R. Lian, I. Arel, H., S. Chen, "A world survey of artificial brain projects, Part II: Biologically inspired cognitive architectures", Neurocomputing, Volume 74, Issues 1–3, pp. 30-49, 2010

[15] F. Bellas, R. J. Duro, A. Faiña, D. Souto, "MDB: Artificial Evolution in a Cognitive Architecture for Real Robots", IEEE Trans. on Autonomous Mental Development, vol. 2, pp. 340-354, IEEE Press, 2010

[16] B. Santos-Diez, F. Bellas, A. Faiña, R.J. Duro, "Lifelong Learning by Evolution in Robotics: Bridging the Gap from Theory to Reality", Proceedings EAIS 2010, pp. 48-53, 2010.