HAL
open science

Distributed Real Time Neural Networks In Interactive Complex Systems

Matthieu Lagarde, Pierre Andry, Philippe Gaussier

To cite this version:

Matthieu Lagarde, Pierre Andry, Philippe Gaussier. Distributed Real Time Neural Networks In Interactive Complex Systems. 2008. hal-00356681

HAL Id: hal-00356681
https://hal.science/hal-00356681

Preprint submitted on 28 Jan 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed Real Time Neural Networks In Interactive Complex Systems

Lagarde Matthieu
ETIS CNRS/ENSEA/Univ Cergy-Pontoise
F-95000 Cergy-Pontoise, France
lagarde@ensea.fr

Andry Pierre
ETIS CNRS/ENSEA/Univ Cergy-Pontoise
F-95000 Cergy-Pontoise, France
andry@ensea.fr

Gaussier Philippe
ETIS CNRS/ENSEA/Univ Cergy-Pontoise
F-95000 Cergy-Pontoise, France
IUF
gaussier@ensea.fr

ABSTRACT

In this paper, we present two graphical softwares which help the modeling and the simulation of real time, distributed neural networks (NNs). Used in the frame of the development of control architectures for autonomous robots, these softwares allow a real time control and an on-line learning of different behaviors. We present as an illustration two control architectures: the first one allowing a mobile robot to navigate using on-line learning of visual places, and the second one allowing a robotic head to learn to express a given set of emotions during imitation games.

Categories and Subject Descriptors

F.2 [**Analysis of Algorithms and Problem Complexity**]: Miscellaneous; I.2 [**Artificial Intelligence**]: Distributed Artificial Intelligence; I.6 [**Simulation And Modeling**]: Model Development

General Terms

Dynamical systems, biological inspired models

Keywords

real time, distributed computing, artificial neural networks, robotics

1. INTRODUCTION

The projects *Leto* and *Promethe* aim at facilitating the development of large neural networks (NNs) with a soft computing approach [5].

Leto is a graphical software dedicated to the design of NNs. Most of the time, a single network is composed of many different groups of neuron (layers, or sub-populations with different dynamics and/or different learning rules). In order to be able to implement such models, *leto* (Figure 1) allows to manage groups of neurons (sizing, creation, suppression, tuning of parameters) and the different types of links connecting these groups. Figure 2 shows the different links that can be built between the neurons of two groups. Note that any kind of linking, such as reentrant connections, is possible.

Promethe is the NN simulator, which schedules the execution of the groups (the activity of the neurons) and the learning (the weight of the links between the groups). A network that has been designed using *Leto* can be run using *Promethe*. Moreover, several *Promethe* can be connected with special input/output groups of neurons, in order to form a distributed NN.

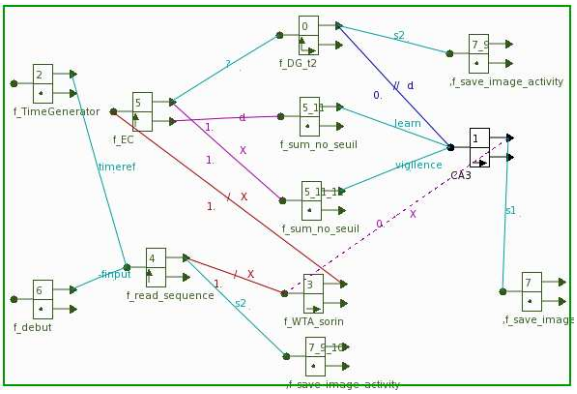

Figure 1: Window of *Leto* allowing building NNs

With these tools, the development of NNs doesn't rely on a procedural language, but on graphic programming. NNs are designed with *Leto* which facilitates the integration of several neural groups, and executed by *Promethe* in order to model the interaction of the groups (see Figure 3 for a global illustration of the development the NNs). Most of the time, we use the NNs as control architectures for autonomous robots as well as learning systems processing large databases (for example in the frame of image processing). In both cases, the level of the simulations and the complexity of the models require the distribution of the networks over several processing units, with important constraints : the repartition must preserve the network stability (information flows, dynamics, learning) and the robot control implies the respect of the time constraints linked to the captors and effectors of the system. Therefore, it is necessary to have suitable tools to

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CSTST 2008 October 27-31, 2008, Cergy-Pontoise, France
Copyright 2008 ACM 978-1-60558-046-3/08/0003 ...$5.00.

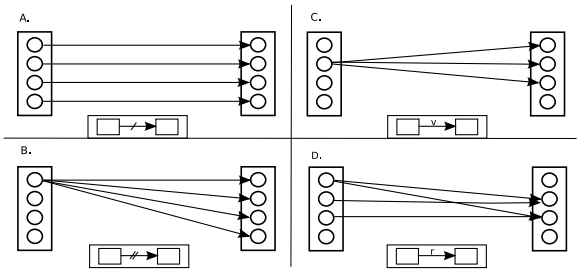

Figure 2: A. one to one links. B. one to all links (only from one neurons is represented for clarity) C. one to neighborhood links with a neighborhood of 1 (only from one neuron is represented for clarity). D. Random links.

develop distributed NN models and to carry them out on robots.

Since their creation, the *Leto* and *Promethe* projects have enormously evolved, following the different technologies of computer science and, in particular, the field of real-time distributed computation.

The first parallelizing architecture used the Parallel Virtual Machine (PVM) interface [10]. This virtual machine enabled to use a pool of workstations as a set of processing units on which different tasks could be assigned. Message passing instructions were used in the code in order to define which kind of message should be sent (or received) by which task. In addition, synchronization mechanisms were available (blocking and unblocking reception for instance). The use of PVM was adequate for small architectures. Nevertheless, there was a unique message queue whatever the number of tasks. This queue turned out to be a bottleneck in the performances of the system. Therefore, a system with a queue for each task communication was preferred.

To distribute an application means to execute it on various computers. Computers can have different processors, different versions of operating system etc.... It is important that the resulting behaviors of the NN applications must not change from a computer to another. Consequently, *Promethe* is able to manage real-time processing in the NNs. There are two different real-time processing with different constraints. The first is the hard real-time: the systems must respect strict temporal constraints. In this kind of applications, if one service does not respect the deadlines, the consequences can be critical. The second is the soft real time, where the temporal constraints exist, but they are applied on the mean of the total time. Hence, if the constraints are not respected punctually, the system can be caught up with on the next step. In the field of interactive complex systems, the issue of real time plays an important role at several levels. This issue is strongly linked to the distribution of the algorithm.

In this paper, we present how a NN is distributed on processing units and/or on several computers. Next, we present the early integration of real time processing in a model. Finally, we show different kind of applications created with *Leto* and simulated with *Promethe*.

2. DISTRIBUTED NEURAL NETWORKS

In the field of epigenetic robotics, our interest is to study the emergence of higher level behaviors, from the combination of low level building blocks. These building blocks are designed as perception-action (PerAc, [6]) loops, each loop being a combination of a reflex pathway and a categorization pathway (it can be seen as a reliable improvement of the counter-propagation principle [8], suitable for the control of autonomous systems). Following the idea that the same "brain" should be able to adapt to very different tasks, and therefore that the same loops can be shared by very different architectures, the distribution of an application between the different loops plays an important role, allowing to reuse a loop in order to test its implication in different emerging behaviors. For example, the same NN loop allowing to extract

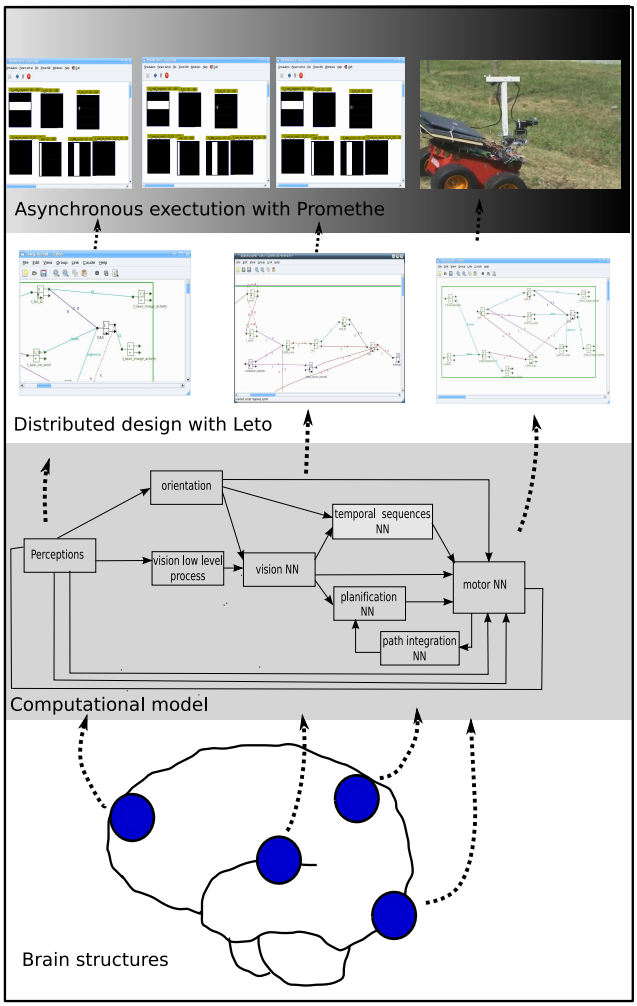

Figure 3: Development of large NNs in order to control autonomous robots. Our design starts from the modeling of brain structures (visual areas, hippocampic loop, functions of the cerebellum, etc). Each structure is assimilated as a set of neural functions and plays a role (learning, filtering, processing, etc) in the whole computational model. From this model, the sets of neural functions (also called neural groups) are distributed on several NNs, each NN being designed using *Leto*. During the robotic experiment, each NN is run on one computing unit using *promethe*

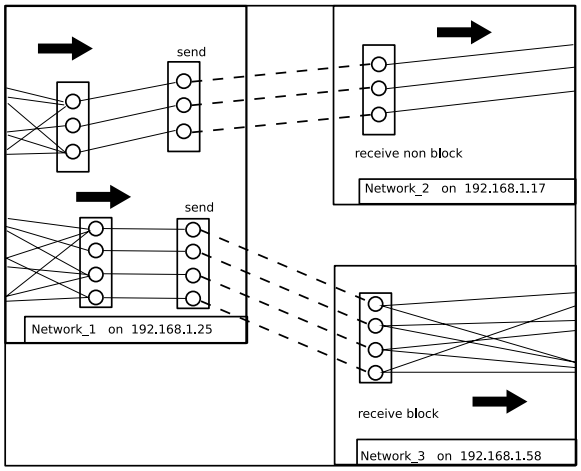

Figure 4: Illustration of a possible communication between several NNs. Communication between network_1 and network_2 is asynchronous, and communication between network_1 and network_3 is synchronous. Send and receive groups have the same number of neurons and the configuration file, or the MetaLeto application solve the links between the name of the network and the IP address of the processing unit.

focus points from the pictures of a CCD camera is used in expression recognition and in the place building mechanism of navigation systems.
To execute an application on several processing units, the different NNs need to communicate. To do so, the simulator *Promethe* integrates a network layer based on BSD sockets with the use of TCP/IP. A set of special groups of neurons allows to send and receive their activities through the network. The message passing protocol can be either synchronous or asynchronous using blocking or unblocking reception mechanisms, in a similar way as PVM does. Figure 4 shows an example of communication between several NNs.

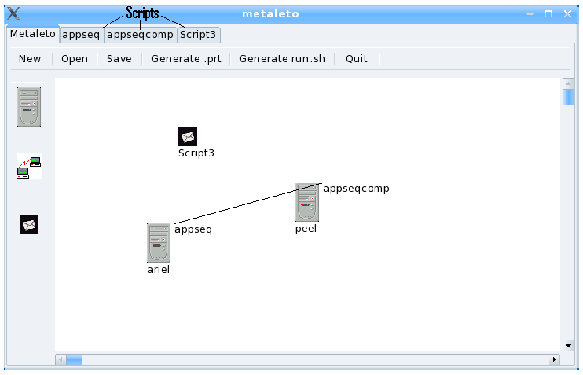

Figure 5: Window of *Metaleto* allowing distributing NNs on computers

But to define which NN communicate with the others, it is necessary to give to the simulator the network address of the others. This part of the distributed software is manually done with the writing of a network configuration file. This file contains 2 sections: the declaration of each NN with the association between his logical name with its network address and its network port. The second section names the network links between each NN. This file allows to *Promethe* to know on which processing unit each NNs is running.
But a distribution of an application defined manually leads to various undesirable behaviors like network dead locks. To facilitate this part of the work, the project *Metaleto* allows the programmer to graphically distribute the various NNs on different computers with simple drag' n drop of NNs on the desired computers. Thanks to the links between NNs, this project can automatically generate the network files. *Metaleto* integrates also *Leto* project to directly create and edit NNs in the same application. Figure 5 shows an application distributed on several computers and an opened NNs.

3. REAL TIME NEURAL NETWORKS

Of course, it is not useful to systematically parallelize all the parts of the same NN. One must balance communication and computation time. As explained in the previous section, each NN of the application is often composed of one or several loops representing the functional unit of the model that we wish to test. Moreover, in the frame of robotic control, some parts of the neural architecture are critical and must deliver a reliable and predictable signal with respect of the time constraints. As a result, we keep in the same NN several branches of neural groups that we wish to execute concurrently. To schedule the execution of these groups

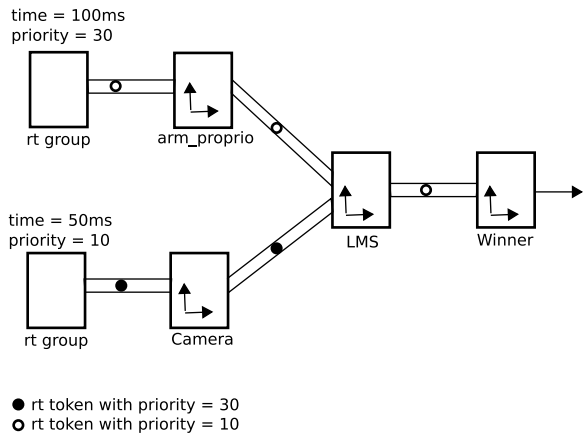

Figure 6: Illustration of the scheduler mechanism of the groups with real time tokens

inside the same network, *Promethe* uses a mechanism of token. At the beginning, the simulator creates one thread by neural group. These threads are destroyed only at the end of the simulation. The scheduler (i) starts the execution of all groups (i.e. all threads) having their entire token. To start a group, it is necessary to have as many tokens as of primary input links (a recursive link is secondary, because as a primary link there is a blockade situation.). When a group has finished, (ii) it gives the token to its successor and (iii) it indicates its termination. These three steps constitute a computation wave repeated during the simulation. In order to save calculation time, the updating of neurons can cut a group into several threads of "sub groups" of neurons. These threads are created and destroyed at each updates.

This scheduling policy results in a cyclical movement of the information where each group is synchronized by the execution of its predecessors. Of course, it also results in a "best effort" policy, where the CPU spends as time as needed to execute the predecessors. To face the real time constraints, we have adapted the scheduler and added a particular type of token: the "real time token". These tokens are generated by a particular group (named "rt group") at a periodic timing. In the general use, a rt group is created at the beginning of a branch. In order to avoid particular problems (figure 6), priorities are assigned to the rt tokens. Consequently, if a group receives two rt tokens from two branches, then the group runs at the timing of the highest priority. When the time is expired, the rt group resets its entire token in all neural groups. This reset can not interrupt the calculations in progress, but stops the computation wave. A second method exists where the computation wave is not stopped. In this case, the developer considers that the next computation wave will respect the time constraint. But this second method starts a new wave even if the previous is not finished. Consequently, it is possible to have two waves in parallel.

4. MERGING ASYNCHRONOUS INFORMATION

Using the distribution and the real time tokens, we obtain parallelized control archtectures whose main advantage is the possibility to test the concurrence and/or the cooperation of many sensori-motor loops in the emergence of the robotic behaviors. Of course, since our models control a single robot, it is important to have a design approach that avoid the potenial bottlenecks when the information of each network have to be merged. Typically, problems arise when many different loops running concurrently are providing abondant and different information about the motor control of the robot. Moreover, since emerging behavior is our major interest, we do not use apriori solutions fixing or selecting the priority of the loops (no subsumption architeture [3]). We prefer to use solutions promoting an intrinsic selection of the action according to dynamical equations simulating a continuous field of neurons. This Neural Field (NF) [1] codes the action space (most of the time the motor space of the robot, like the angles of joints to control, or the orientation of mobile robots, etc.). In addition, a readout mechanism [11] allow to directly extract from the NF's activity the speed command to send to the corresponding joint [2]. Thanks to the dynamical properties of the NF, noisy inputs are filtered, concurrent information is merged (cooperation) or selected (competition) simply according to the bifurcation capabilty of the field. Finally, the intrinsic memory of the system allows a stable robot control under the condition that the NF respects the time constraints of the device. Such a system allows reliable decision making with asychronous loops

5. APPLICATIONS

5.1 Indoor Navigation

Robot navigation is a problem that has been largely studied by the scientific community during several decades. In our work in this domain, models are composed of several processes each participating to the global behavior emergence.

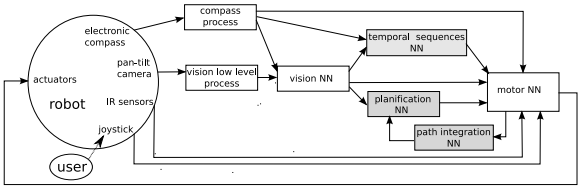

Figure 7: Illustration of an application of indoor navigation composed of several NNs distributed and executed in parallel. The motor NN is composed of a dynamic neural field allowing the decision making between the "joystick", "infra-red sensors" and the place recognition provided by "vision NN".

For instance, we know how to teach the robot a specific navigation behavior [7] from simple elements(figure 7):

- The compass process reads the electronic compass value and sends it to the visual and motor neural networks.
- The low level vision process transforms visual inputs into usable informations to characterize and learn visually different places.
- The vision neural network allows the place recognition and binds it to the action the robot has to realize in that place (sensory-motor learning). Therefore, it sends motor commands to the motor script. The motor neural network uses a neural field in order to select the action the robot will execute. It has to choose between, in increasing order of priority, the motor command coming from the vision NN, the motor command coming from the joystick (user) and the motor command coming from the obstacle avoidance (infra-red sensors).

Following the same principle, adding other process to the model will modify the global emergent behavior the robot will execute. For instance, by adding a planing process (dark grey box), the system is able to map his environment (topological map) and thus to define a strategy to reach a specific resource [4]. In ongoing works, a new temporal sequence learning process [9] (light grey box) tends to be add to the model. This process allows a memorization of a specific task as a sequence of predictable states. The question is : which properties are going to emerge from the interaction of the model with these supplementary processes.

5.2 Expressions Recognition

This application consists in the online learning and the recognition of facial expressions with a robotic head. The experimental protocol is the following: in a first phase of interaction, the robot produces a random facial expression (sadness, happiness, anger, surprised) plus the neutral face during 2s. Between each facial expression, the robot returns to a neutral face to avoid human misinterpretations of the facial expression from the robot during 2s. The human subject mimics the robotic head. In this phase of interaction, the robot learns the facial expressions. If the robot displays the happiness then the subject mimics it. Once this first phase is finished (after 5 to 10 min, according to the subjects "patience"), the generator of random emotional states is stopped in order to begin the second phase of interaction.

Now, the robotic head is able to mimic the human's facial expression. In the learning phase, the robotic head displays

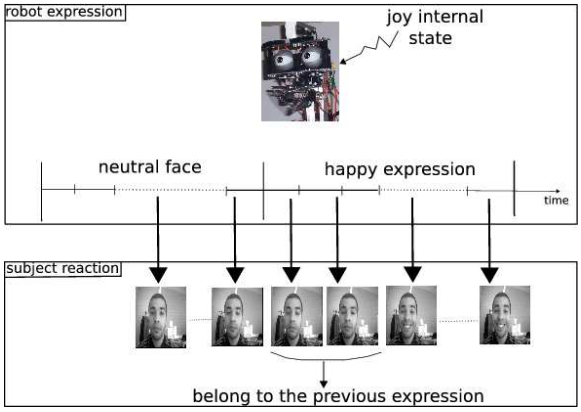

Figure 8: Phase shifting between the human facial expression and the robot facial expression during an imitation game (the human imitating the robot).

each facial expression during 2s. This time can disturb the learning because if the robot learns the first images which are still associated to the human previous facial expression then the previous expression is unlearned (figure 8). The presentation time of a given expression must be long enough to neglect the first images. We must be careful, if this time is too long, the subject might lose his concentration.
In our system, time constraints are crucial and have to be respected by the NNs.

6. CONCLUSION

We have presented softwares that allow creating distributed NN models managing real time processing. The global behavior of a system emerges from the processes executed in a parallel way. The distribution of NNs is defined a priori by the developer thanks to *Metaleto* program. The execution under real time constraints is managed by *Promethe*. These programs allow graphically developing distributed and real time NNs and propose a global view of them. The main parameters (time constraints or how cut the NNs) are defined by the developer. But, it is always possible to have various problems like time constant too short to execute a NN, or dead lock after a wrong distribution of the NNs. In future improvements, we can imagine to add new tools like dead locks checking in *Metaleto*. Therefore with the same software, the NNs could be automatically cut without explicit network groups in the NNs. The distribution of NNs raises important questions: is the behavior of one NN the same that a similar NN distributed on several computers? Each part of NN is scheduled by one scheduler of each *Promethe*. A possible evolution could be using a single virtual scheduler for all NNs. We can imagine each scheduler running each NN communicate the tokens the ones with the others (figure 9). This way would also allow a global managing of real time processing on the entire NN and not on each part. This raises the question : do all the NNs have to be scheduled in real-time with a global scheduler? For example, in order to improve the calculation of "vision NNs" which consume very large amount of CPU time, it is interesting to get several pictures in one wave. Consequently, this particular NN can

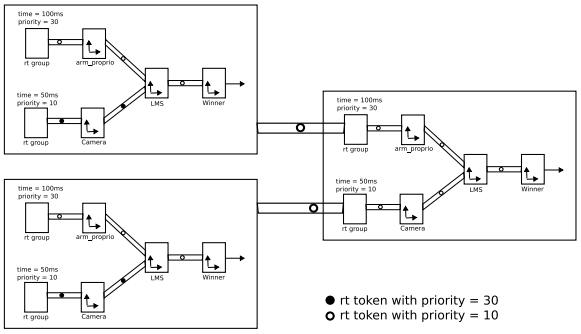

Figure 9: Illustration of a possible global distributed scheduler with "rt tokens" passing from a scheduler of a *Promethe* to another

not be scheduled with others. Hence, the use of a unique global scheduler would not be suitable. Finally, in the context of emergent behaviors, the management of distributed and real-time processes is not trivial. For different kinds of distribution and/or for different kinds of parameters in real-time processing, the global behavior of the system can be different.

7. AKNOWLEDGEMENTS

This work is supported by the French region Ile de France, the Institut Universitaire de France (IUF) and the European project Feelix Growing : FP6 IST-045169.

8. ADDITIONAL AUTHORS

Additional authors: Boucenna Sofiane (`boucenna@ensea.fr`), Giovannangeli Christophe (`giovannangeli@ensea.fr`), Cuperlier Nicolas (`cuperlier@ensea.fr`), Maillard Mickael (`maillard@ensea.fr`) and Quoy Mathias (`quoy@ensea.fr`)

9. REFERENCES

[1] S. Amari. Dynamic of pattern formation in lateral-inhibition type by neural fields. *Biological Cybernetics*, 27:77–87, 1977.

[2] P. Andry, P. Gaussier, and J. N. and. B.Hirsbrunner. Learning invariant sensory-motor behaviors: A developmental approach of imitation mechanisms. *Adaptive behavior*, 12(2), 2004.

[3] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.

[4] N. Cuperlier, M. Quoy, C. Giovannangeli, P. Gaussier, and P. Laroque. Transition cells for navigation and planning in an unknown environment. In *The Society For Adaptive Behavior SAB 2006*, pages 286–297, Rome, 2006.

[5] P. Gaussier. *Simulation d'un système visuel comprenant plusieurs aires corticales: Application à l'analyse de scènes.* PhD thesis, University of Paris Sud Centre d'Orsay, december 1992.

[6] P. Gaussier, S. Moga, M. Quoy, and J. Banquet. From perception-action loops to imitation processes: a bottom-up approach of learning by imitation. *Applied Artificial Intelligence*, 12(7-8):701–727, Oct-Dec 1998.

[7] C. Giovannangeli and P. Gaussier. Autonomous vision-based navigation: Goal-oriented action planning by transient states prediction, cognitive map building, and sensory-motor learning. In *Proc. of the 2008 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2008)*, Nice, France, 2008.

[8] R. Hetch-Nielsen. Applications of counterpropagation networks. *Neural Networks*, 1(2):131–139, 1988.

[9] M. Lagarde, P. Andry, and P. Gaussier. The role of internal oscillators for the one-shot learning of complex temporal sequences. In J. M. de Sa, L. A. Alexandre, W. Duch, and D. Mandic, editors, *Artificial Neural Networks – ICANN 2007*, volume 4668 of *LNCS*, pages 934–943. Springer, 2007.

[10] M. Quoy, S. Moga, P. Gaussier, and A. Revel. Parallelization of neural networks using PVM. *Lecture Notes in Computer Science*, 1908:289–296, 2000.

[11] G. Schöner, M. Dose, and C. Engels. Dynamics of behavior: Theory and applications for autonomous robot architectures. *Robotics and Autonomous Systems*, 16(4):213–245, 1995.