EVOLVING PID-LIKE NEUROCONTROLLERS FOR NON-LINEAR CONTROL PROBLEMS

S. Doncieux J.-A. Meyer
LIP6 - Animatlab
8, rue du capitaine Scott
75015 Paris
{Stephane.Doncieux,Jean-Arcady.Meyer}@lip6.fr

Abstract

This article describes an empirical approach to non-linear control problems that calls upon the evolution of modular neural networks. This approach may be bootstrapped with modules that encode knowledge stemming from linear or non-linear control theory, and it seems to be applicable to non-stationary problems as well. It has been applied here to the control of the trim and altitude of a simulated lenticular blimp that was subjected to several perturbations. The corresponding results demonstrate the superiority of the evolved networks over a hand-designed controller. They also demonstrate the capacity of evolution to exploit the intrinsic non-linearities of artificial neurons in order to generate different solutions, likely to be adapted to the context of the considered application.

Key Words: evolution, neural networks, modules, lenticular blimp

1. Introduction

Real world systems often exhibit non-linear and non-stationary behaviors that deeply challenge traditional control techniques of engineers like PIDs. These techniques usually rely on an analysis of a model of the system to be controlled and on an inversion around a given operating point of a linearized version of this model. As a consequence, the corresponding controllers remain effective only as long as the linearized model represents the system's actual behavior.

Although several research efforts have been devoted to the extension of these mathematical approaches to non-linear and non-stationary systems [1, 2], they do not yet seem to have converged to any general conclusions, according to which a given method would clearly seem more appropriate to design the controller of a given system.

Insofar as the corresponding choice is still largely empirical, there is room for other approaches to control that, although they are less mathematically sound than those of traditional engineering and do not lead to any stability or convergence theorem, still might compete with respect to the effectiveness and efficiency of the controllers they generate, at least for given systems and in given conditions. Among such empirical approaches, those that call upon the evolutionary design of neural networks look promising because they have already been applied to complex systems [3, 4, 5] and because, as will be shown below, they have the capacity of capitalizing on traditional PID controllers and of being applicable to non-linear and non-stationary domains.

This point of view will be illustrated here in the case of a highly non-linear system: a simulated lenticular blimp that has to be maintained as horizontal as possible at a given altitude, despite various perturbing factors. It will be shown that efficient controllers may be built without providing any other information on the system to be controlled than a fitness function that assesses its performance without exploiting any knowledge on its underlying mathematical model. Instead, this function calls upon mere state variables that could be monitored by off-the-shelf sensors, thus leaving to evolution the goal of discovering on its own how to control the system.

The article first describes the simulation

model and the control task. Then it describes how ModNet, the evolutionary framework that was used in this work, may generate modular neural controllers and how the corresponding process may be bootstrapped with traditional P, I and D modules. Finally, the results that have been achieved on the considered control task will be presented and their significance will be discussed. These experiments have been implemented thanks to the SFERES framework described elsewhere [6].

2. The control task

Although the ultimate goal of our research is to control a lenticular blimp 10 meters-wide with a rich sensorimotor equipment, the objective of the present work was to control the trim and altitude of a simulated version of this aircraft, calling upon two inclinometers and one altimeter, on the one side, and with three motors, on the other side (fig. 1). The task to be achieved was to maintain the blimp as horizontal as possible at a given altitude, a difficult control task because of non-linear couplings between the engine's trim and altitude and because of the perturbing effects that were imposed by the experimenter on the wind speed and direction, as well as on the blimp's trim and altitude.

The simulation model that was used is more than 3000-instructions long and is too complex to be described here. However, some relevant details are given in [7, 8].

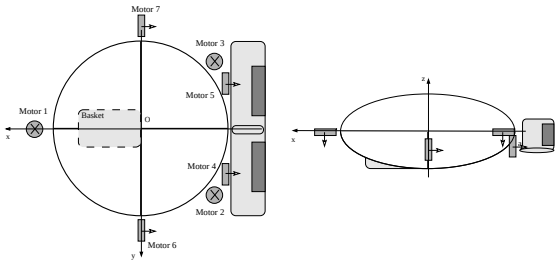


Figure 1: Two views of the lenticular blimp. In the experiments reported here, only motors 1 to 3 were used.

3. The ModNet evolutionary framework

Classical approaches to the evolution of neural controllers [5] call upon neurons and connections that they assemble to generate whole networks. The problem with this approach is that evolution usually has to start from scratch and that it is difficult to use some a priori knowledge to speed up convergence. Therefore, to tackle difficult problems, an incremental approach is frequently used, according to which a human designer decomposes the overall control task and evolves sub-controllers for each sub-task he has identified. He then has to compose the whole controller by assembling the sub-controllers, an endeavor that is far from easy.

ModNet proceeds differently. Instead of manipulating neurons and connections, it operates on modules. These modules are small neural networks that may be spontaneously discovered by the evolutionary process. They may also stem from an initial pool of templates provided by the experimenter either because, having been generated during previous evolutionary runs, they might be useful to solve the current problem, or because they encode some form of expert knowledge.

ModNet was previously applied to generate controllers for a cartpole [8, 9]. In those experiments, evolution was not provided with initial templates and derivative modules - a functionality essential to solving the problem. On the contrary, they were spontaneously discovered and exploited by the evolutionary process[1].

3.1 Chromosomes

A chromosome in ModNet is made up of three components: a list of model-modules, a list of modules and a list of links (fig. 2).

The list of model-modules codes both the organization and the inner parameters (e.g., connection weights) of the modules that will appear in the final neural network. The next two lists describe the structure of the network. They indicate how modules are linked together and how they are linked to the network's inputs and outputs. The list of modules specifies, for each module appearing in the neural network, whose

[1] These modules have indeed inspired the derivative modules we have used in experiments described in this article.

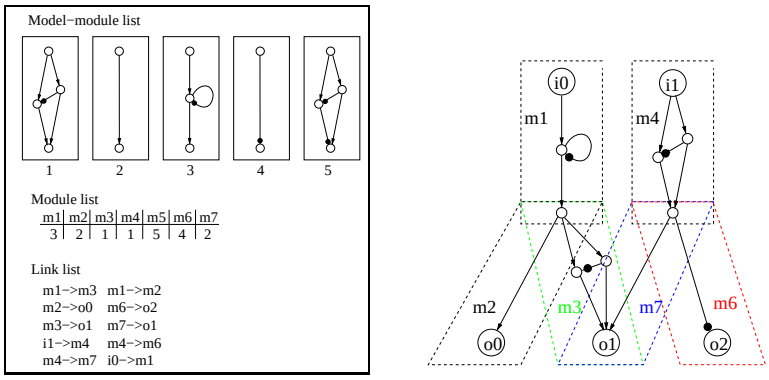


Figure 2: Left: An example of a chromosome manipulated by ModNet. The chromosome is made up of three components: a list of model-modules, a list of modules and a list of links. m_l are modules, i_n are sensory neurons, and o_m are motor neurons. Right: the corresponding decoded neural network. Modules $m3$ and $m4$ are copies of the same model-module 1. Likewise, modules $m2$ and $m7$ are copies of model-module 2. Connections ending with a dot or an arrow are respectively negative and positive connections

model-module it is a copy of. The same model-module may then appear at different places in the neural network, thus reducing the amount of information required to describe the controller. Symmetries may also be easily handled this way.

The list of links specifies how modules are interconnected in the network. Instead of specifying a connection, a link concerns a module output (or a network output) and a module input (or a network input) that will be merged in the final network. For example, because a link associates the output of module 1 to the input of module 2 in the network of fig. 2, the output neuron of module 1 is merged with the input neuron of module 2.

3.2 Genetic operators

There are two categories of mutation operators in Modnet. In the first category, these operators modify parameters in the neural network[2] that are coded as 8-bit binary strings. In the second category, mutation operators change the network's structure because they act on each of a chromosome's three components. Thus, model-modules may be added to, or removed from, the model-module list, while the other two lists may be modified by inserting or deleting modules in the chromosome.

Because, in Modnet, mutations can't modify the internal structure of modules, fragile structures are protected from destructive operations. Likewise, because crossover operators exchange model-modules between individuals, a useful structure can easily and quickly propagate to newborn individuals, as demonstrated in [8, 9].

3.3 ModNet's bootstrapping procedures

Each model-module is generated from an initial pool of templates provided by the experimenter. Depending upon the chosen setting, it is a mere copy of a randomly chosen template, or it recombines the template's parameters while keeping their structure, or it recombines both the structure and the parameters. Be that as it may, the first option makes it possible to bootstrap the evolutionary process by providing templates that seem, for whatever reason, to be useful for solving the considered problem. For instance, it is thus possible to provide artificial evolution with templates that connect given sensory inputs to given motor outputs, thus avoiding a random search throughout the pattern of all possible connections. Likewise, the initial pool can be seeded with templates that perform elementary computations which probably need to be instantiated in the final network. However, it must be emphasized that evolution is not committed to stick to such initial templates: they only serve to provide the first generation with a priori interesting solutions. Subsequently, during the course of the evolutionary process, initial

[2] In the experiments described here, the only parameters concerned were connection weights.

templates may well get lost through mutations if they turn out not to be as useful as expected, or they may be transformed into more efficient modules.

4. Application to blimp control

To evolve neural controllers for the blimp, a dedicated version of ModNet was used. In particular, an initial population of 10,000 individuals was randomly generated, out of which only the best 100 were allowed to evolve further. The selection algorithm was rank-based, with 40% of the individuals being replaced by new ones at each generation. Finally, specific templates and a dedicated fitness function were used in the experiments reported on here.

4.1 Templates

Three different templates, liable to approximate respectively the proportional, integrate and derivative building blocks of standard control methods, seeded the initial pool (fig. 3). The P module merely implements a direct connection from its input to its output. The D module is likely to compute a derivative by subtracting the value of an input signal received at time step $t-1$ to the value of the same signal received at time step t. Naturally, this will occur only provided the signs of the corresponding connection weights are appropriately set, i.e., when their product is negative. As to module I, it can integrate a signal approximately when the weight of the self-recurrent connection is positive.

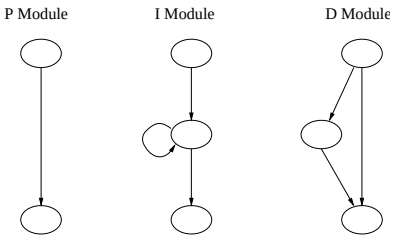


Figure 3: Modules P, I and D. P module is a proportional module. I module affords the ability to integrate a given signal, and D module can approximate a derivative, provided connections have appropriate weights.

In the experiments described here, only the structures of the templates were fixed. Their weights had to be evolved along successive generations.

4.2 Fitness function

To evaluate individuals and select those that were allowed to propagate their genetic material to the next generation, their behavior was assessed over a given evaluation period of 10,000 time steps, corresponding to 250 simulated seconds - i.e., a period long enough to check that a steady-state behavior with no subsequent drift was obtained. A *viability zone* [10, 11] of ± 0.35 rad was defined for pitch and roll angles, so that every controller that allowed these *essential variables* move outside this range of values was considered non-viable. Finally, the following fitness function was used:

$$f(x) = p(x) + \frac{1}{nb_{DOF}} \sum_{i \in DOF} \left(1 - \frac{\sum_t (d_i(x,t)^2)}{T}\right)$$

The first term, $p(x)$, measured the percentage of the evaluation time that was spent before the blimp occasionally moved outside the viability bounds, and the second term, lower than one, measured the quality of the control as it quantified the mean deviation between a given degree of freedom (DOF) and its target value ($d_i(x,t)$ being its instantaneous value). nb is the number of DOFs concerned, and T is the evaluation length. The controlled DOFs were pitch and roll, whose target values were permanently set to 0 rad, and altitude, whose target value could change over time. Thus, the first objective of the fitness function was to force the evolutionary process to generate controllers that kept the blimp inside its viability zone; a second objective was to improve the efficiency of these controllers. Additional capacities of robustness were afforded by the specific conditions in which the controllers were evaluated. In particular, the simulated blimp was subjected to many abrupt perturbations during the evaluation period, according to which the orientation and amplitude of the wind, the blimp's current pitch and roll values, and its target altitude were set to new, randomly chosen values. These perturbations occurred at randomly chosen instants, on an average rate of three times per evaluation period.

4.3 Reference controller

In the perspective of comparing the empirically evolved controllers with more traditional engineering solutions, a reference controller calling upon three PID modules that independently managed the blimp's three degrees of freedom was hand-designed. The input of each such PID module was the error value of the degree of freedom it was supposed to maintain, while its output was sent to an interface that set the three motor thrusts by summing the contribution of each controller. Thus, the altitude error generated identical thrusts at each motor, the roll error generated thrusts of opposite signs for motors 2 and 3, and the pitch error generated thrusts of opposite signs for motor 1, on the one hand, and for motors 2 and 3, on the other hand. The three inner parameters of each PID module were also optimized with an evolution strategy [12, 13] using the same fitness function as described above.

5. Experimental results

Not all the neural networks that were thus generated retained the P, I and D-like modules that served to bootstrap the evolutionary process. However, these modules were empirically recombined and integrated into larger controllers in ways that extended their utility for non-linear regimes of the simulation model, as illustrated below in the case of two specific networks.

5.1 Exploitation of non linear couplings

All the efficient controllers that were obtained in the last generations of the evolutionary process did exploit non-linear couplings between pitch and altitude. These couplings stem from the facts that, thanks to its rear fin, the blimp quickly moves to face the wind and that, thanks to its lenticular shape, it tends to behave like a wing. Thus, if its nose slants at a negative angle with respect to a facing wind[3], the blimp moves very quickly upwards. Likewise, if it faces the wind with a positive angle, it soon moves down.

The important point to emphasize is that such couplings were not taken into account, either in the pool of templates that were provided to the evolutionary process, or in the fitness function that was used - simply because the human experimenter had not foreseen that they would be important. Actually, the hand-designed controller that independently managed each of the blimp's three DOFs did not exploit such couplings either, and turned out to be three times slower than the best controller generated by evolution [7] (fig. 4).

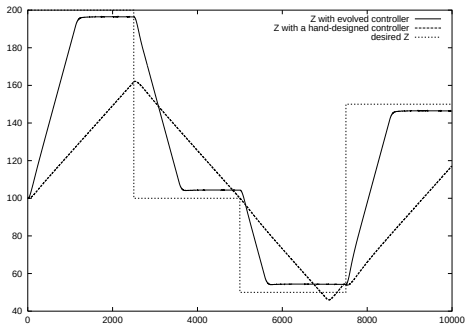


Figure 4: Comparison of the behaviors generated by an evolved neural network and by a hand-designed system with respect to altitude control. The evolved controller is three times faster than the hand-designed one. A time step corresponds to 25ms; altitudes are given in meters.

The exploitation of non-linear couplings by an evolved controller is clearly demonstrated on fig. 5. When the blimp falls below the desired altitude (between time-steps 0 and 1500, for instance), the blimp has to go up. The controller keeps the pitch around a negative non-zero value (near -0.2 rad) that allows the blimp to exploit the wind and quickly climb. Likewise, when the blimp is above the desired altitude, the pitch is kept around 0.2 rad to allow the blimp to rapidly descend.

5.2 Evolved control strategies

Not all the networks evolved implemented the same control strategy. In particular, some of them generated some variety of 'bang-bang' control, while others generated more continuous behaviors. However, controllers in both categories exhibited similar efficiencies in terms of the fitness function that was used.

[3] By convention, trim angles are considered positive when the aircraft leans towards the ground

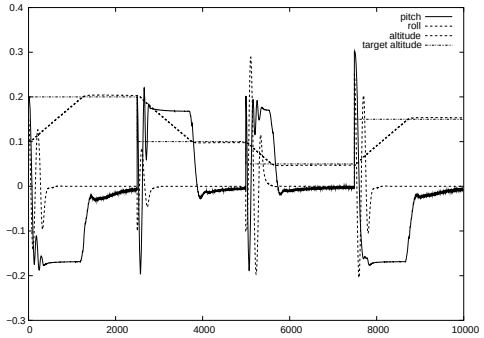


Figure 5: Example of behavior generated by an evolved neurocontroller. Angles are in radians and altitudes in meters/1000. A time step corresponds to 25ms.

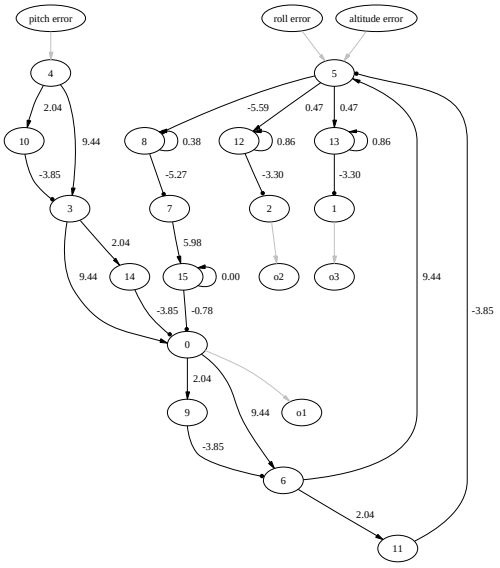


Figure 6: Neural network 1, which implements a bang-bang control strategy. Connections ending with a dot or an arrow are respectively negative and positive connections. Grey connections only concern sensors or effectors (they have a fixed weight of 1). For reasons of clarity, individual modules are not distinguished. However, it so happens that neurons 5, 12 and 2 are included in a I module, and that neurons 4, 10 and 3 belong to a D module, for instance.

5.2.1 Bang-Bang controllers

Neural network 1 (fig. 6), generated after 1000 generations, controls trim and altitude using a 'bang-bang' strategy during the phases where the blimp's altitude is near the target value (fig. 7).

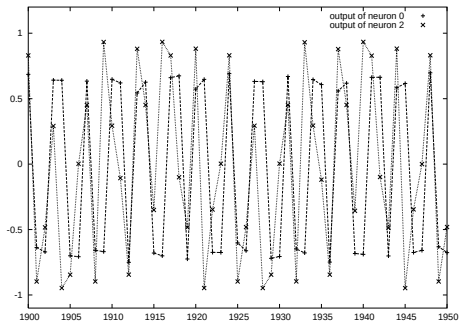


Figure 7: Outputs of neurons 0 and 2 in neural network 1. The output of neuron 2 exhibits a simple bang-bang behavior: a succession of $+a$ and $-a$ values, a being constant. The output of neuron 2 is a succession of lines whose slopes approximately equal $+ka$ and $-ka$, k being constant too.

This strategy is possible thanks to the recurrent loop between neurons 5, 8, 7, 15, 0, 9, 6, 11 and 5. If the signs of all the corresponding connections are multiplied, a negative value is obtained[4]. This means that the network tends to bring the activity of a neuron in the loop to a negative (resp. positive) value when it is positive (resp. negative), thus creating the oscillation required by the bang-bang strategy. Commands to motors 2 and 3 are generated through an I module that tends to integrate the signal, as demonstrated on fig. 7.

This behavior is inhibited by high values of the altitude error, i.e., when the blimp has to climb or go down. Motors 2 and 3 are then used at their maximum power to accelerate the blimp's ascension or descent.

A close look at the inner organization of neural network 1 further reveals that evolution did find a tortuous way to manipulate the functionalities of the templates it was provided with. Instead of directly incorporating a P module into the controller, it used an I module (neurons 7, 15 and 0 of the neural network on fig. 6) in which it set the weight of its recurrent connection to 0.

Although the corresponding bang-bang strategy is very efficient in simulation, it would

[4]Provided connections between neurons 0 and 9 and 9 and 6 - which are small relative to the connection between 0 and 6 - are neglected, and provided the same thing is done with connections involving neurons 6, 11 and 5.

not be applicable with real motors that would have a hard time surviving such a discontinuous regime. Fortunately, competing controllers generating a more continuous behavior were also discovered by evolution.

5.2.2 Continuous controllers

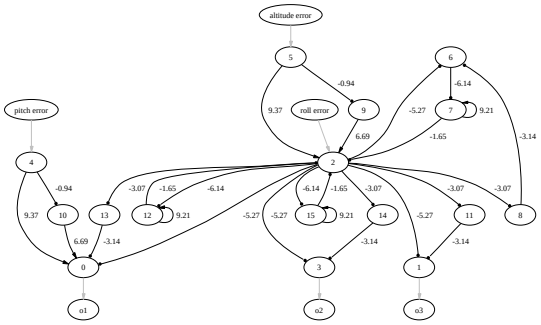
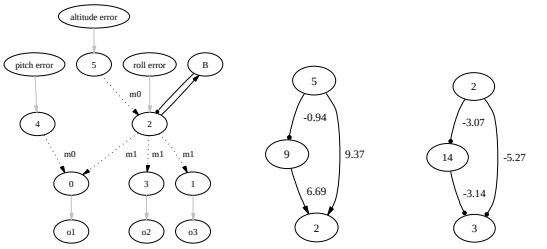


Figure 8: Neural network 2, which implements a continuous control strategy. Connections ending with a dot or an arrow are respectively negative and positive connections. Grey connections only concern sensors or effectors. Individual modules are not distinguished. However, it so happens that neurons 6, 7 and 2, for example, are included in an I module, and that neurons 2, 14 and 3 belong to a D module.

Neural network 2 (fig. 8), generated after 2000 generations, is one of the controllers that implemented a continuous control strategy. Several I and D modules within that network use neuron 2 as both input and output. They may all be replaced by a single neuron "B", as shown on fig. 9. This neuron acts as a bias whose output is constant throughout an experiment. The value it takes on is determined by the initial conditions, and can be either -1 or $+1$. This structure departs from symmetry in the control of altitude and allows the blimp to go up faster than down, or the opposite, depending on the initial conditions. Interestingly, this result is due to some sort of *overfitting process* according to which evolution generated solutions that specifically fit the particular conditions in which the fitness was evaluated.

Other modules have been used as expected. For instance, the D module m1 in fig. 9 generates a behavior in the linear domain that approximates a derivative behavior, as can be seen on fig. 10 after the 3500^{th} time step, where the



Simplified network module m0 module m1

Figure 9: Simplified representation of neural network 2 on the left. Recurrent connections have been replaced by an equivalent "bias" neuron "B" whose output is constant throughout an experiment: it equals -1 or $+1$, depending on the initial conditions. Dotted lines correspond to modules m0 and m1, which are described on the right.

module removes the constant part of the input signal. But evolution has set the parameters of this module so that it is able to generate another behavior in the saturated domain. Rather than exhibiting a zero output, this module produces a saturated output when its input is saturated, as can be seen on the same figure before the 3500^{th} time step. Thus, the evolutionary process discovered a way to adapt the initial functionality of this module so that it behaves efficiently in two different conditions: when altitude is near the target value (linear case) and when it is far from it (saturated case).

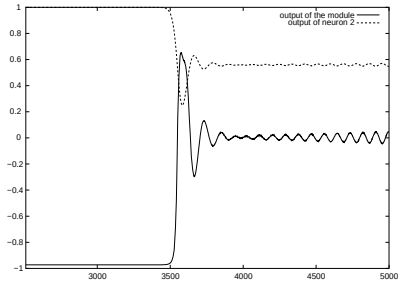


Figure 10: Time variations of the module's m1 input (neuron 2) and output (neuron 3). Two different behaviors are exhibited: a constant maximum output when the input is saturated, and a derivative-like behavior when the input is not.

6. Discussion

In this work, evolution was provided with the standard functionalities of linear control theory - through P, I and D-like modules - and applied to a challenging control problem, where a complex non-linear dynamic system had to be controlled in frequently perturbed conditions. The evolutionary process kept some of these modules as they were, diverted some others from their initial functionalities, and merged them all in augmented structures that proved to be able to cope with non-linearities.

Although the possibility that better solutions to this challenging problem will be produced via some well-founded engineering approach to non-linear control still needs to be assessed, the empirical solutions that were discovered here by an evolutionary approach turned out to perform better than, at least, one traditional solution based on linear PIDs alone. Likewise, the possibility that many such empirical approaches, compared over many different application problems, might trigger the sort of generalizations that could boost non-linear control theory must be assessed. Conversely, any progress of this theory might be useful for bootstrapping the evolution with other and more efficient initial modules.

Because the P, I and D-like modules that were used in this application were composed of standard artificial neurons, their behavior was non-linear and only approximated P, I or D functionalities in the linear region of the neural transfer functions. It has been shown here that these non-linearities have been exploited by evolution to implement different control strategies, like bang-bang or continuous. Which one was ultimately implemented in a given controller depended on mere chance, through the random effects of the genetic operators. Other exploitations of the non-linearities of individual neurons could probably be discovered in other experiments and other contexts.

Providing neural network modules to evolution allows domain knowledge to be exploited, while leaving evolution the choice of doing it or not. It is an alternative to the incremental approach in the search for scalability. Another advantage is that we know what the modules provided are supposed to do and can exploit this knowledge in our analysis of the networks generated. The breakdown into modules furthermore provides information on the overall structure of the network and makes it easier to understand, as we can analyze the behavior of each module separately before studying them together in the whole network.

Finally, it should be stressed that evolutionary approaches, because they deal with a population of solutions, are a priori well adapted to the management of non-stationary control problems [14]. Thus, a given solution, although possibly less efficient than another at solving the current version of a given problem, may well be fit enough to be maintained in the population. When circumstances change, and when a modified version of the problem has to be solved, this solution may turn out to be the best one, or to constitute a good stepping-stone on the route to this optimal solution. An extension of this idea, which manages a permanently updated model of a given dynamic system, has been applied within the framework of *anytime learning* [15]. It could be implemented in the context of the evolutionary approach advocated here, thus extending to non-stationary problems the range of solutions that have been applied here to non-linear cases.

7. Conclusion

This work describes an empirical approach to non-linear control problems that calls upon the evolutionary generation of modular neural networks. This process may be bootstrapped with modules that encode knowledge stemming from linear or non-linear control theory, and it seems to be applicable to non-stationary problems as well. It has been applied here to the control of a simulated lenticular blimp that was subjected to several perturbations. Neural controllers that efficiently solved the control task did capitalize on P, I and D-like modules that they combined in a more efficient way than the one that was conceived by a human designer. Likewise, interesting adaptive capacities were afforded to the evolutionary process through the possibility of exploiting the non-linear transfer function of individual neurons, some modules being able to generate a given behavior in the linear domain and another behavior in the non-linear domain.

References

[1] A. Isidori. *Nonlinear Control Systems.* Springer-Verlag, Berlin, 2nd edition, 1989.

[2] J.-J. Slotine and W. Li. *Applied Nonlinear Control*. Prentice Hall, 1990.

[3] Sutton Miller and Werbos, editors. *Neural Networks for Control*. MIT Press, 1990.

[4] A. M. S. Zalzala. *Neural Networks for Robotic Control: Theory and Applications*. Ellis Horwood, 1996.

[5] J.-A. Meyer. Evolutionary approaches to neural control in mobile robots. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, San Diego, 1998.

[6] S. Landau, S. Doncieux, A. Drogoul, and J.-A. Meyer. Sferes: un framework pour la conception de systèmes multi-agents adaptatifs. *Technique et Science Informatique*, 21(4), 2002.

[7] S. Doncieux and J.-A. Meyer. Evolving neural networks for the control of a lenticular blimp. In G. R. Raidl et al., editor, *Applications of Evolutionary Computing, EvoWorkshops2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, EvoSTIM*. Springer Verlag, 2003.

[8] S. Doncieux. *Évolution de contrôleurs neuronaux pour animats volants : méthodologie et applications*. PhD thesis, LIP6/AnimatLab, Université Pierre et Marie Curie, Paris, France, 2003.

[9] S. Doncieux and J.-A. Meyer. Evolving modular neural networks to solve challenging control problems. In *Proceedings of the Fourth International ICSC Symposium on ENGINEERING OF INTELLIGENT SYSTEMS (EIS 2004)*, 2004. (to appear).

[10] W. R. Ashby. *Design for a Brain: The Origin of Adaptive Behavior*. Chapman and Hall, 1952.

[11] J.-A. Meyer and A. Guillot. Simulation of adaptive behavior in animats: Review and prospect. In Meyer and Wilson, editors, *Proceedings of The First International Conference on Simulation of Adaptive Behavior*. The MIT Press, 1991.

[12] T. Bäck and H. P. Schwefel. Evolution strategies i: Variants and their computational implementation. In *Genetic Algorithms in Engineering and Computer Science, Proc. First Short Course EUROGEN-95*. 1995.

[13] H. P. Schwefel and T. Bäck. Evolution strategies ii: Theoretical aspects. In *Genetic Algorithms iEngineering and Computer Science, Proc. First Short Course EUROGEN-95*. 1995.

[14] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[15] A. C. Schultz and J. J. Grefenstette. Continuous and embedded learning in autonomous vehicles: Adapting to sensor failures. In *SPIE Int. Symposium on Aerospace/Defense Sensing, Simulation and Controls (AeroSense 2000)*, 2000.