Evolution of neurocontrollers for complex systems: alternatives to the incremental approach

Stephane Doncieux Jean-Arcady Meyer
Animatlab - LIP6, France
http://animatlab.lip6.fr
{Stephane.Doncieux,Jean-Arcady.Meyer}@lip6.fr

Abstract

Applications of neural networks to solve challenging control problems still face important difficulties when scalability issues are involved. Usually, such difficulties are tackled according to an incremental approach that consists in decomposing a given task into simpler sub-tasks that may be separately solved. In this article, we describe two alternatives to this approach and demonstrate their efficiency on the control of a simulated lenticular blimp, i.e., a complex dynamic platform with five sensors and seven motors.

1 Introduction

Evolving neural networks is an approach that proved to be efficient at controlling a variety of dynamic systems [7, 9, 4, 5]. Moreover, when a given system turns out to be too complex for a controller to be directly evolved, an incremental approach often leads to satisfactory results. In this case, the overall task is decomposed by a human designer into simpler ones for which separate controllers are evolved and then recombined into a unique network. In particular, this strategy worked on a Khepera robot [8] confronted with a "survival" task decomposed into several sub-tasks, like obstacle avoidance and battery recharge. It also worked on a 6-legged animat [6] whose survival depended upon its capacities to walk, to avoid obstacles, and to follow an odour gradient. Nevertheless, in many applications, there are no clues about how to decompose the initial problem, or about how to recombine the partial solutions into a coherent whole. As for the hopes that co-evolutionary approaches [3] will help automating this process, they have yet to materialize.

In this paper, we present two alternatives to this incremental strategy, which also exploit available knowledge, but avoid decomposing the initial problem. The first one consists in changing the building blocks that evolution manipulates: instead of neurons and connections, evolution deals with modules each representing a given sub-network. The second one consists in providing the evolutionary process with clues about which connections are likely to be useful in the final controller. In both cases, the knowledge thus provided serves only to bootstrap evolution. The corresponding information is only exploited to orient the search during the initial random generation. Later on, it can be altered or forgotten under the effects of genetic operators.

We implemented these tools in ModNet, a framework dedicated to the evolution of neural networks [1], and we used them to generate neural controllers for a lenticular blimp, a complex dynamic platform with five sensors and seven motors. Although this research aims at controlling a real blimp, results presented in this paper are preliminary and call upon a realistic simulation.

2 ModNet

In traditional approaches to the evolution of neural networks, entities like neurons or connections are manipulated by the evolutionary algorithm in a way that forbids passing to the next generation complex structures that were discovered and proved to be useful in the current one. On the contrary, ModNet affords this possibility to evolution because the units that are manipulated are modules that describe sub-networks whose structures are globally stable throughout the course of evolution and that serve as building blocks on which the evolutionary process may capitalize. These modules may encapsulate some a priori knowledge about the problem to be solved, or they may emerge from the evolutionary process.

Every chromosome associated with ModNet is made up of three components: a list of model-modules, a list of modules and a list of links between modules (figure 1). The model-modules list contains the description of each modules that can be used in the final network. The list of modules makes a connection between a module number and the corresponding model. This makes it possible to reuse a single module structure in several different places in the network. The last part of the chromosome is a list of links that connect modules to each other or to the network's inputs and outputs.

Both mutation and crossover operators are used in ModNet. Mutations may affect a model-module, by changing its structure[1] or parameters[2]. Other mutations may alter the structure of the network by randomly changing either the list of modules or the list of links. Crossovers exchange model-modules between individuals.

Each model-module that is included in the model-module list, either at initialization time or through mutations during the course of evolution, is drawn from a pool of modules that is initially provided by the experimenter. This set of initial modules may be chosen at random or it may integrate some knowledge - stemming from an a priori engineer's analysis or from results of former experiments - which is instantiated in module structures or in parameter values. In particular, it is possible to specify the structure of a module, i.e., which neuron is connected with which neuron, but not the corresponding connection weights. Likewise, it is possible to specify these weights, or to simply specify the sign of some specific connections within the structure. Be that as it may, any such initial specification may be later overcomed by mutations during successive generations: its sole purpose is to bootstrap the evolutionary process.

Additional bootstrapping knowledge may be taken into account in ModNet through the use of so-called "connectivity patterns". Indeed, it is possible to a priori specify how some modules could be connected to each other or to the network's inputs and outputs. Again, such initial patterns may be exploited and propagated from generation to generation or they may get lost.

3 Control of a lenticular blimp

The goal of this application is to keep a lenticular blimp (figure 2) above a given visual target, at a given altitude and as horizontal as possible.

Inputs to the evolved controllers are the pitch[3] and roll[4] angles, the altitude and the relative position of the target in the frame relative to the blimp[5]. The outputs of the controllers are sent to the blimp's seven motors. Details about the simulation model and results concerning separate controls of the pitch, roll and altitude are to be found in [2].

We performed three series of 30 evolutionary runs, each involving a population of 100 individuals and 500 generations. These runs differed by the initial pool of modules we provided (figure 3). The first series of runs served as a control experiment: no connectivity pattern was provided and the only module included in the initial pool was made of a single input, a single output and a direct connection linking

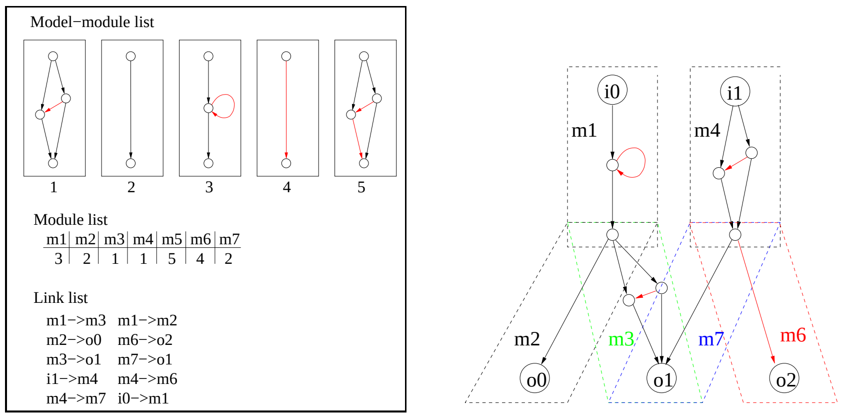
[1]This feature is not used in the results reported here.

[2]Every parameter characterizing the developed network - like a connection weight or the slope of a transfer function - is encoded in these model-modules.

[3]rotation along the lateral axis of the blimp.

[4]rotation along the longitudinal axis of the blimp.

[5]On the real platform, this information is given by a visual tracking system.



Module list

m1	m2	m3	m4	m5	m6	m7
3	2	1	1	5	4	2

Link list

m1−>m3	m1−>m2
m2−>o0	m6−>o2
m3−>o1	m7−>o1
i1−>m4	m4−>m6
m4−>m7	i0−>m1

Figure 1: Left: An example of chromosome generated by ModNet. The chromosome is comprised of three components: a list of model-modules, a list of modules and a list of links. 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.

them. This module was called a P module, because its output was proportional to its input, at least in the linear domain of the transfer function. In the second series, we included P modules - but with one or two outputs - in the initial pool again, and we provided a connectivity pattern to orient the search. Such pattern reflected a priori engineering knowledge about which sensor should be connected to which motor [1], as well as results of careful analyses of some networks evolved during preliminary runs not mentioned herein. In the third series, the same connectivity pattern was used again, but additional modules with one or two outputs - respectively called D and I modules -, which were designed to afford capacities for both derivative and integral computations, were added to the previous P modules in the initial pool. The D modules could compute from an ongoing signal the difference between two successive time steps. The I modules called upon an intermediate neuron with a self-recurrent connection, a structure that is capable of integrating a signal, provided the corresponding connection weights are appropriately set.

The fitness function we used is the sum of two terms:

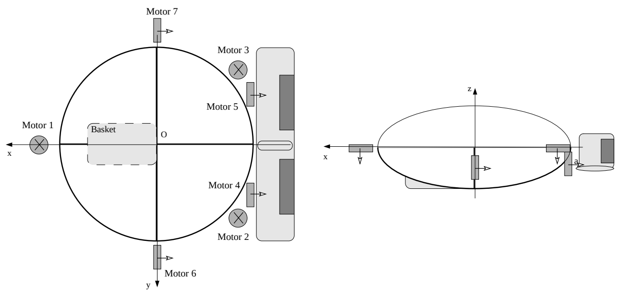


Figure 2: The lenticular blimp and its seven motors.

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

The first term, $p(x)$, measures the percentage of the maximum evaluation time the blimp spent before overstepping the viability domain of the blimp - which was set to ± 0.7 rad for pitch and roll values [6]. The second term measures the performance of the control as the average, on each degree of freedom

[6]The limits of the viability domain are defined by experts and characterize values beyond which the blimp is considered as no more controllable. No constraints were imposed to the blimp's altitude and position.

Generation	**Data**	**wo CP**	**w CP**	**w CP and PID**
20	Maximum	100.775	100.815	100.745
	Average	100.617	100.708	100.655
	Minimum	100.493	100.626	100.602
100	Maximum	100.798	100.818	100.82
	Average	100.674	100.757	100.693
	Minimum	100.529	100.645	100.608
500	Maximum	100.809	100.824	100.873
	Average	100.700	100.779	100.742
	Minimum	100.532	100.654	100.633

Table 1: Fitness values obtained in four series of experiments on the control of the blimp. Thirty runs were made for each condition. “wo CP”: runs without a connectivity pattern, “w CP”: runs with a connectivity pattern, “PID”: runs with Derivative and Integral modules.

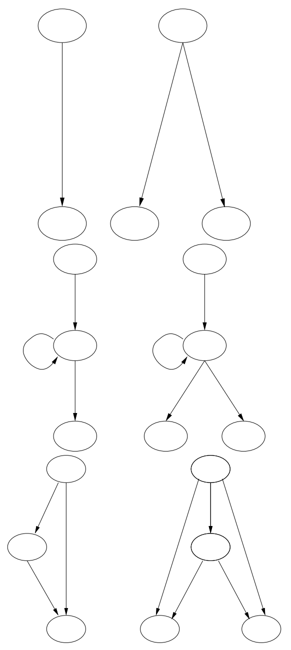


Figure 3: Examples of P (top), I (middle) and D (bottom) modules, with one or two outputs, that were used to bootstrap the evolutionary runs.

and over the whole evaluation period, of the square distance between actual and target positions. Target values were set to 0 for the pitch and the roll, and to a given value, likely to be changed over time, for the altitude and the horizontal position. This term is normalized in order to lie between 0 and 1 (we use a constant of 20m to normalize the horizontal position and a constant of 100m for the altitude). Thus, the total fitness varies between 0 and 101. A value greater than 100 means that the blimp did not overstep the viability domain during the entire evaluation period, and a value of 101 means that it always stayed at the vertical of the target point meanwhile.

The results of the three series of runs are summarized in Table 1) and exhibit statistically significant (Kruskal-Wallis test, prob < 0.05) differences. Thus, bootstrapping the evolutionary process with connectivity patterns and dedicated modules seems to constitute a valuable alternative to the incremental approach. Moreover, a closer look at the results reveals important differences in the controlled behaviors obtained in the three series. In particular, the best controllers obtained with the control experiments (wo CP) efficiently keep the pitch, the roll and the horizontal position, but none of them is able to keep the altitude as well. On the contrary, the best controllers generated in the two other series of runs are able to control all the DOFs together.

The most efficient controller obtained during these experiments (figure 4) belongs to the third series (w CP and PID) and generates the behavior reported on figure 5. Interestingly, although I modules were provided in the initial pool, this controller doesn’t call

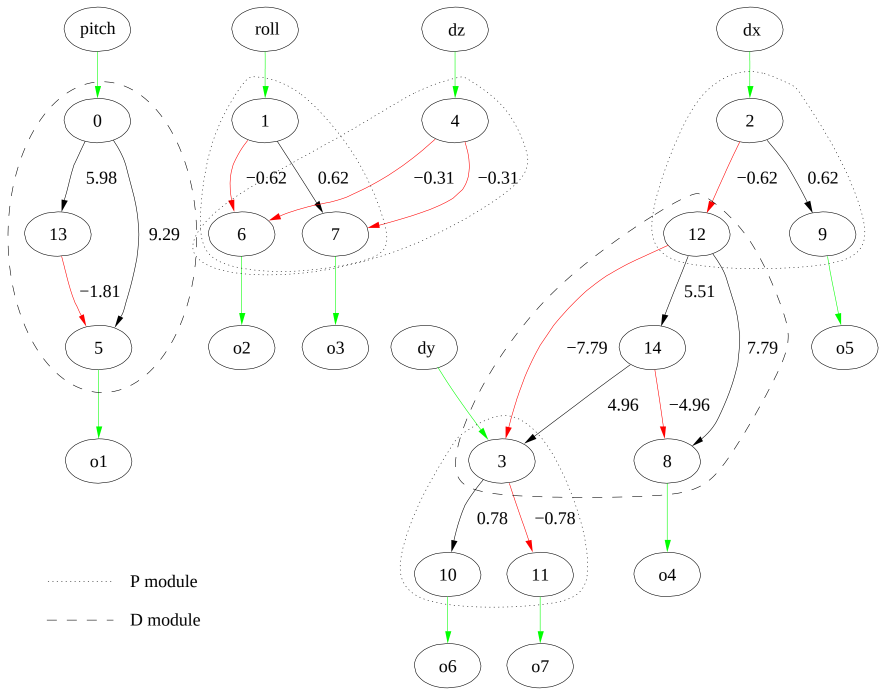


Figure 4: Best neural network generated after 500 generations in a run of the third series (w CP and PID). The functionalities of the corresponding modules are singularized and inner connection weights are given.

upon such modules and relies on P and D modules only. This probably means that the functionalities afforded by integral computations are somehow compensated by the inner workings of the whole network, or that such functionalities would be useful only for bridging the gap between the fitness of this controller and the maximum possible fitness.

Finally, the fact that the average fitness values of the runs of the second series (w CP) is greater than that of runs of the third series (w CP and PID) suggests that bootstrapping the evolutionary process with both connectivity patterns and dedicated modules may be advantageous - because better fitnesses may occasionally be obtained - but at the risk of enlarging too much the corresponding search space and of slowing down evolution. In particular, if I modules are not mandatory, suppressing them from the initial pool could help optimizing the experimental setup. Additional and systematic experiments might help clarify this point.

4 Conclusion

This article describes two alternatives to the incremental approach to the neural control of complex dynamic systems. They both amount to bootstrap the evolutionary process with some information about the kind of neural network that seems appropriate to solve the considered problem. Results that have been obtained on a simulated lenticular blimp suggest that these procedures may help automatically designing controllers for systems with numerous sensors and actuators. However additional experiments are needed to better assess the generality of such conclusion.

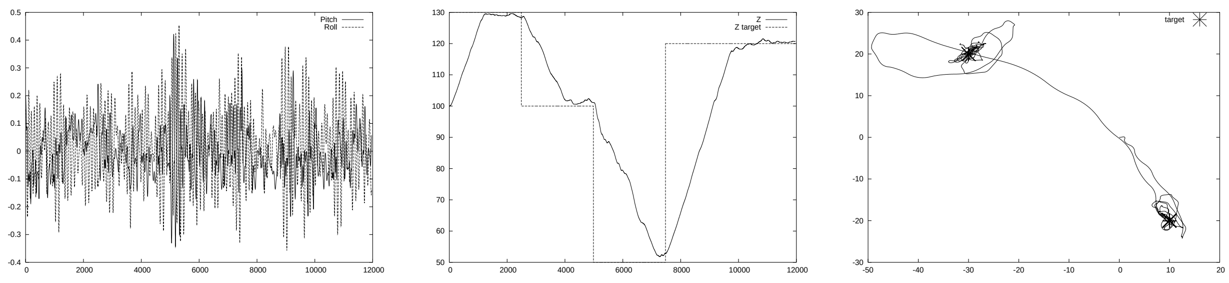


Figure 5: Behavior of the blimp controlled by the network of figure 4. Left: pitch and roll in radians (x axis in 25 ms time steps). Middle: altitude in meters. Right: 2D plot of the horizontal trajectory. During the evaluation period (25 sec), the coordinates of the visual target as well as the target altitude are changed in respectively one (at the 5000th time step) and three occasions (at the 2500th, 5000th and 7500th time step). Meanwhile, the wind direction is changed several times, hence the observed oscillations around equilibrium values.

References

[1] S. Doncieux. *Évolution de Contrôleurs Neuronaux pour Animats Volants : Méthodologie et Applications.* PhD thesis, Université Paris 6, 2003.

[2] 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.

[3] D. Floreano, S. Nolfi, and F. Mondada. Competitive co-evolutionary robotics: From theory to practice. In R. Pfeifer, B. Blumberg, J.-A. Meyer, and S. W. Wilson, editors, *From Animals to Animats: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior.* MIT Press, 1998.

[4] F. Fogelman-Soulie and P. Gallinari, editors. *Industrial Applications of Neural Networks.* World Scientific Publishing Co, 1998.

[5] Y. H. Kim and F. L. Lewis. *High-Level Feedback Control with Neural Networks.* World Scientific, 1998.

[6] J. Kodjabachian and J.-A. Meyer. Evolution and development of neural networks controlling locomotion, gradient-following, and obstacle-avoidance in artificial insects. *IEEE Transactions on Neural Networks*, 9:796–812, 1997.

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

[8] J. Urzelai, D. Floreano, M. Dorigo, and M. Colombetti. Incremental robot shaping. *Connection Science Journal*, 10(384):341–360, 1998.

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