Behavioral Repertoire Learning in Robotics

Antoine Cully
ISIR, Université Pierre et Marie Curie-Paris 6,
CNRS UMR 7222
4 place Jussieu, F-75252, Paris Cedex 05,
France
cully@isir.upmc.fr

Jean-Baptiste Mouret
ISIR, Université Pierre et Marie Curie-Paris 6,
CNRS UMR 7222
4 place Jussieu, F-75252, Paris Cedex 05,
France
mouret@isir.upmc.fr

ABSTRACT

Learning in robotics typically involves choosing a simple goal (e.g. walking) and assessing the performance of each controller with regard to this task (e.g. walking speed). However, learning advanced, input-driven controllers (e.g. walking in each direction) requires testing each controller on a large sample of the possible input signals. This costly process makes difficult to learn useful low-level controllers in robotics.

Here we introduce BR-Evolution, a new evolutionary learning technique that generates a behavioral repertoire by taking advantage of the candidate solutions that are usually discarded. Instead of evolving a single, general controller, BR-evolution thus evolves a collection of simple controllers, one for each variant of the target behavior; to distinguish similar controllers, it uses a performance objective that allows it to produce a collection of diverse but high-performing behaviors. We evaluated this new technique by evolving gait controllers for a simulated hexapod robot. Results show that a single run of the EA quickly finds a collection of controllers that allows the robot to reach each point of the reachable space. Overall, BR-Evolution opens a new kind of learning algorithm that simultaneously optimizes all the achievable behaviors of a robot.

Categories and Subject Descriptors

I.2.6 [**Computing Methodologies**]: Artificial Intelligence—*Learning*; I.2.9 [**Computing Methodologies**]: Artificial Intelligence—*Robotics*

General Terms

Algorithms

Keywords

Evolutionary Algorithm, Evolutionary Robotics, Mobile Robotics, Behavior, Exploration, Novelty Search

GECCO'13, July 6–10, 2013, Amsterdam, The Netherlands.

1. INTRODUCTION

Learning is one of the keys to make autonomous robots able to cope with unforeseen situations, whether on a remote planet or, more prosaically, in a house. In particular, learning low-level sensorimotor behaviors is essential to allow robots to pursue their mission in all situations, for instance after mechanical damage [3]. This learning is typically tackled with policy gradient methods [11, 14, 24] or evolutionary search [9, 13, 20, 4, 15], because these algorithms are well suited for reinforcement learning in continuous space.

The authors of these algorithms show that new behaviors can be learned in 20 minutes [24] to dozens of hours [9], depending on the openness of the search space and the technique used. However, in most of these studies, they considered controllers that are learned for a single instance of the studied behavior. For example, when investigating walking controllers, they often designed their algorithm to make the robot walk *on a straight line* and *at the fastest possible speed*.

This problem is much simpler than the general problem of low-level controller learning: *learning a general controller that can accept commands issued by a higher-level system (e.g. a planning algorithm)*. This problem is typically addressed by testing controllers with several different inputs and averaging the rewards for all the tested scenarios (e.g. [13, 20]). For instance, Mouret et al. [20] used an evolutionary algorithm to design a neural network that pilots a simulated flapping robot; they tested the ability of the neural network to drive the robot to 8 different targets and the fitness function was essentially the sum of the distances to the targets.

Unfortunately, this approach is very costly. First, it tests each candidate solution in each scenario, thus increasing the learning time by at least an order of magnitude. Second, learning a general controller is much more challenging than learning a specialized one. This challenge may even be out-of-reach for many robotics applications because it requires more advanced algorithms – which may not yet exist – and long learning times – which may be unfeasible with a robot.

An alternative is to learn a *repertoire of simple controllers* instead of a single, general controller. This method avoids the challenge of learning a complex controller; however, it typically involves as many learning processes as there are behaviors in the repertoire. When evolving a gait controller, this means launching the evolutionary algorithm for each possible target point, hence slowing down learning by a factor equal to the number of targets.

In the present paper, *we introduce a new technique, named*

Behavioral Repertoire Evolution (BR-Evolution), that allows a robot to learn a whole repertoire of simple controllers with a single run of an evolutionary algorithm. The proposed algorithm is substantially faster than the previously described approaches and it can be applied to any genotype (neural networks, programs, parameters, ...) as well as most control tasks.

Our technique is based on a simple observation: candidate solutions that are discarded because they have a low fitness score could often have been used in the controller collection. In the walking robot example, controllers that make the robot move along a straight line are often explicitly rewarded. However, controllers that make a robot turn right are actually equally interesting – but they correspond to a different behavior in the behavioral repertoire of the robot. Since these controllers are discarded or badly rewarded, they are "wasted".

Our main insight is that these wasted controllers can be exploited to evolve a behavioral repertoire. Nonetheless, while interesting, the controllers that lead to different behaviors are not all equal. Some of them are more "degenerated" than others. To select the best variants, we take inspiration from the recently introduced "novelty search with local competition"[17] that looks for a set of different "species" that covers the morphological space while applying an intra-species selective pressure. We evaluate our algorithm by evolving a behavioral repertoire for a simulated, hexapod robot that must be able to go forward, backward and turn in both directions. We compare results to the separate evolution of each behavior of the repertoire.

2. BACKGROUND

2.1 Learning Low Level Controllers

A low level controller (LLC) is the element that drives the agent by sending commands to the motors. We separate $LLCs$ in two categories: inputs-driven $LLCs$ and un-driven $LLCs$. For instance, an un-driven LLC can command a primitive action [22] like "stand up", "step", "turn right", "turn left" and so on. An inputs-driven LLC can execute several primitive actions or parametrized actions like "turn 30 degrees right", "step 10 meters forward", according to the received inputs. These LLCs are typically combined with high-level controllers [22], which control the global behavior of the agent using various decision and planning algorithms [5, 6].

The majority of studies dealing with LLC learning proposes methods for un-driven $LLCs$. This generates limited behavioral repertoires because the agent is only able to do one thing, for example straightforward walking. The methods usually employed are policy gradient methods [11, 14, 24] or evolutionary algorithms [9, 13, 20, 4, 15], which optimize the controller's performances according to a desired action. They were successfully applied to several domains, from snake crawling [18] to bipedal walking [24].

Comparatively few works deal with inputs-driven LLCs, for example the work from Mouret et al [20], previously mentioned, or the one from Kodjabachian [13] on the control of an hexapod robot with a neural network. These methods lead to larger behavioral repertoires, but they are still limited by the learning method: they require to test the controllers on each situation [20] or to use an incremental evolution process [13]. All of this significantly increases both the learning time and the difficulty of the learning problem.

2.2 Many Un-Driven LLCs

One alternative to learning inputs-driven $LLCs$ is to consider a collection of un-driven $LLCs$. Thus, a high level algorithm will select the right controller according to the desired action instead of sending the appropriate instruction to an inputs-driven LLC

In biology, some clues tend to show that the brain works with these two principles in different areas. For example, the Superior Colliculus controls the coordination of eye, head and arm movements during reaching at a single visual target [2]. This complex controller is able to reach visual targets for every head and targets positions. Conversely, it has been shown [8] that primates use different regions of their Cortex for different primitive actions: the excitation of different areas of a monkey's brain makes the animal execute different primitive actions, like grasping or hand-mouth interactions. The set of all these regions can be seen as a collection of un-driven $LLCs$.

Using a collection of controllers can be unnecessarily complex. For instance, in order to change the amplitude of a movement, it can be easier to change the reference instead of learning another controller. But, in certain cases, a collection of simple and specialized solutions can be more efficient. For instance, in supervised learning, *boosting* and *experts mixtures* techniques are based on this principle. Indeed, *experts mixtures* [10] train several primitive experts, each on a different problem, instead of training a complex expert dealing with every situations.

The simplicity and efficiency of controller collections and their possibility to merge $LLCs$ of different types of actions– for example moving controllers with grasping controllers– give us the intuition that it is a promising way to endow a large behavioral repertoire to an agent.

2.3 Novelty Search With Local Competition

A long-standing challenge of artificial life is to craft an evolutionary process that discovers a wide diversity of interesting artificial creatures. While evolutionary algorithms are good candidates for this process, they usually converge to a single species of creatures. To overcome this issue, Lehman and Stanley recently proposed a new process that compares creatures' performances between those with similar morphologies [17]. This process, called *novelty search with local competition*, relies on a multi-objective evolutionary algorithm to combine novelty search [16] with performance competitions between similar individuals.

In this variant of novelty search, two objectives are simultaneously optimized: (1) the novelty objective ($novelty(i)$), which corresponds to the original novelty search algorithm [16], and (2) the local competition objective ($local\_obj(i)$), which compares the performances of each individual ($perf(i)$) to those that share its local niche. This second objective is defined as the number of individuals in the local niche that i outperforms according to the performance criterion $perf(i)$. The evolutionary algorithm thus favors individuals that are new, those that are higher-performing than their neighbors and those that are optimal trade-offs between novelty and "local performance".

Novelty search with local competition was successfully applied to an experiment consisting in evolving morphologies

and controllers for virtual, walking creatures. Novelty search fosters the population to explore new types of morphologies while the local competition rewards creatures that walk faster than morphologically similar creatures. In this case the local competition is useful, because comparing slow behaviors of massive creatures with fast gaits of little ones, could be nonsensical for many applications.

The local competition aims at generating a multitude of functional morphologies. This is a kind of *morphological* repertoire, where several types of creatures execute the same action. This is the opposite of our goal, as we want to get one type of creature executing a multitude of actions; but this algorithm represents a good starting point for our method.

3. BR-EVOLUTION

Collections of un-driven *LLCs* seem to be a promising way to endow a high behavioral repertoire to an agent. But how can the agent learn all these *LLCs*?

A learning algorithm could be launched for each un-driven *LLC*. If a collection contains 100 actions, the same number of learning processes needs to be executed and the learning time will be multiplied by the same factor. This relation between the number of executions and the learning time makes this method very time-consuming.

Another possibility consists in using multi-objectives optimization algorithms, where each objective corresponds to a desired action. This technique is able to simultaneously learn several controllers. Nevertheless, it has been proved [12] that these algorithms can only deal with 3 or 4 objectives. With more objectives they are equivalent to random search. A collection of controllers with only 3 or 4 actions does not represent a large behavioral repertoire.

Here we propose a new approach, called *BR-Evolution*, to simultaneously learn a large number of un-driven *LLCs*. This method is based on the novelty search with local competition [17], but with a completely different goal from the initial use of this algorithm.

The BR-Evolution uses the novelty objective, not to evolve morphologies, but in order to explore the space of possible actions. A controller is considered new when its behavioral result is different from results obtained with previous controllers. This objective allows the BR-Evolution to do the exploration without the discrimination of a fitness function.

The use of the local competition is slightly different from the original implementation. In the original algorithm, it is used to optimize primary objective of the individual, while in the BR-Evolution, it serves to promote a secondary objective among the controllers generating the same action (in the local niche), for example the controller with the best stability among controllers moving 50 cm forward.

A difference with the local competition algorithm is how the archive is used. Originally, it only serves to log all encountered individuals during the evolutionary process and their behaviors. Thanks to it, the algorithm can compute the novelty of an individual compared to what has been seen before. Nevertheless with the initial novelty search, only the first encountered individual will be added in the archive. Next individuals executing the same action will not be saved, even if they are more efficient according to another criterion. This leads to an archive figuring all kind of achieved actions but not the best action for each kind.

In the BR-Evolution approach, the archive, in addition to logging encountered solutions, constitutes directly the collection of controllers and thus the results of the algorithm. Thereby it should constantly contain the best controller for each kind of explored actions. The comparisons between individuals are made according to the local competition criterion.

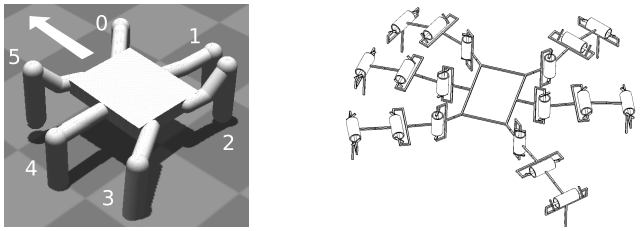

Figure 1: (Left) Snapshot of the simulated robot in our ODE-based physics simulator. The robot lies on a horizontal plane and contacts are simulated. (Right) Kinematic scheme of the robot. The cylinders represent actuated pivot joints.

During the evolutionary process, if an individual of the population is better than the most similar individual in the archive, the two individuals are swapped. This change does not affect the novelty search since individuals from the archive are only replaced by similar ones. At the end of the evolutionary process, the archive contains all the best encountered individuals in all the explored search space.

To summarize, BR-Evolution approach relies on three principles:

- Two objectives are optimized simultaneously:

$$\text{maximize} \begin{cases} \text{Behavioral Novelty}(\mathbf{i}) \\ \text{Local Performance}(\mathbf{i}) \end{cases}$$

- Each individual with a new behavior is saved in the archive;
- When an individual performs better than its equivalent in the archive, they are swapped.

The pseudo-code of the algorithm is presented in Algorithm 1. Our algorithm is based on the same variant of NSGA-II [7] (a Pareto-based multi-objective evolutionary algorithm) as the original implementation of the local competition [17]: instead of using the mechanism that encourages the diversity along each pareto front (crowding factor), we use a mechanism which encourages genotypic diversity.

4. EXPERIMENTS

4.1 Evaluated Scenario

We evaluate our approach on a simulated hexapod robot (figure 1) with a learning gait task. The aim of this experiment is to obtain a collection of un-driven *LLCs* that allows the robot to reach every point in its vicinity. The position reached after running the controller during 3 seconds is considered as the endpoint of the trajectory. The controllers are based on sinusoids. They are described with 24 parameters, each of them having five possible values (Appendix B).

For this experiment, the novelty objective drives the population to explore new reachable endpoints. Its value ($Novelty(i)$, see equation 1) is set as the average distance between the

endpoint of the current controller (E_i) and the endpoints contained in $neigh(i)$:

$$Novelty(i) = \frac{\sum_{j \in neigh(i)} \|E_i - E_j\|}{|neigh(i)|} \quad (1)$$

Where $neigh(i)$ is the set of the 15 individuals $j \in (pop_{controller} \cup archive)$ whose endpoints are the nearest from E_i (according to an Euclidean distance).

To get high novelty values, individuals have to follow trajectories leading to endpoints far from the rest of the population. The population will thus explore all the area reachable to the robot. Each time a new area is explored, a controller able to access to it will be saved in the archive.

However, in order to sequentially execute saved behaviors, a special attention is paid to the final orientation of the robot. Indeed, as the endpoint of a trajectory depends on the initial orientation of the robot, we need to know how the robot ends its previous movement when we plan the next one. Instead of keeping this degree of freedom, we decided to encourage behaviors that end their movements with an orientation aligned with their trajectory. This facilitates the chaining of controllers.

As the type of controllers used with the robot is only composed of periodic signals (see Appendix B), the robot cannot execute arbitrary trajectories. For example, beginning its movement by a turn and then go straight is impossible. If we want the robot to move in different directions, its trajectories would be necessary circular.

We consequently constrain the desired robot's trajectories to portions of circles centered on the lateral axis, with a variable radius (see figure 2A). Forward (or backward) straight trajectories are still possible with the particular case of an infinite radius. This kind of trajectories is suitable for motion control as some complex trajectories could be decomposed in a succession of these circle portions.

To encourage the population to follow these trajectories, the local competition objective is set as the angular difference between the arrival orientation and the tangent of circular trajectory that corresponds to the endpoint ($perf(i) = |\theta(i)|$, see figure 2B and equation 2).

$$\begin{array}{l} Local(i) = |\{X_1, X_2, ..., X_j, ...\}| \\ \text{with} \\ X_j \in neigh(i), perf(X_j) < perf(i) \\ perf(i) = |\theta(i)| = |\alpha(i) - \beta(i)| \end{array} \quad (2)$$

We launched 40 runs of the experiment, the parameters and source code of which are in appendix A. With this experiment, we want to show how the robot is able to *autonomously*:

- discover possible movements;
- cover a high proportion of the reachable space;
- generate a behavioral repertoire.

4.2 Control Experiments

To our knowledge, no work directly tackles the question of learning simultaneously all the behaviors of a controller collection, thus we cannot compare our approach with an existing method. We implemented a straightforward method where the desired endpoints are preselected. A controller will be optimized to reach each wanted point.

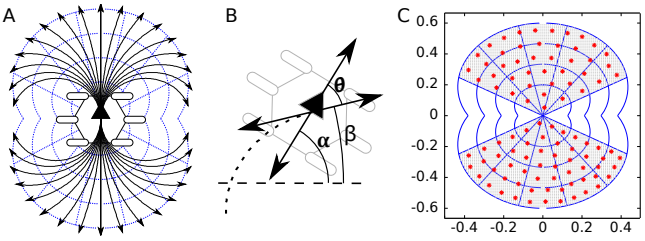

Figure 2: (A) Examples of trajectories following a circle centered on the lateral axis with several radii. (B) Definition of the desired orientation. θ represents the orientation error between α, the final orientation of the robot, and β, the tangent of the desired trajectory. These angles are defined according to the actual endpoint of the individual, not the desired one. (C) Reachable area of the robot viewed from top. A region of interest (ROI) is defined to facilitate post-hoc analysis (gray zone). Its boundaries are defined by two lines at 60 degrees on each side of the robot. The curved frontier regroups all reachable points with a curvi-linear abscissa lower than 0.6 meters. All of these values were set thanks to experimental observations of commonly reachable points. Dots correspond to targets selected for the control experiments.

We define 100 target points, spread thanks to a K-means algorithm [23] over the defined ROI of the reachable area (see figure 2C). We then execute several multi-objective evolutionary algorithms (NSGA-II [7]), one for each reference point. At the end of each execution of the algorithm, the nearest individual to the target point in the Pareto-front is saved in an archive. This experiment is called "nearest" variant. We also save the controller with the best orientation within a radius of 10 cm around the target point and we call this variant "orientation". The objectives used for the optimization are:

$$\text{minimize} \begin{cases} Distance(i) = \|E_i - E_{Reference}\| \\ Orientation(i) = |\alpha(i) - \beta(i)| \end{cases}$$

We launched 40 runs of both variants, the parameters and source code of which are in appendix A. Each algorithm is programmed in the Sferes$_{v2}$ framework [19].

4.3 Results

Resulting behavioral repertoires from a typical run of BR-Evolution and the control experiments are pictured on figures 3 and 6. The reachable space, for every experiment, is sampled, both in front and in rear. However, for the same number of evaluations, the area is less covered with the control experiments than with the BR-Evolution. With only 100 000 evaluations, it is about twice larger with the BR-Evolution than with both control experiments. At the end of the evolution (1 000 000 evaluations), the reachable space is more dense with our approach. With the "nearest" variant of the control experiment, all target points are reached (see figure 2C), this is not the case for the "orientation" variant.

The orientation error is qualitatively more important in the "nearest" control experiment during all the evolution than with the other experiments. This error is important

Algorithm 1 BR-Evolution algorithm (G generations)

$pop_{controller} \leftarrow \{c^1, c^2, \ldots, c^{S_{controller}}\}$ (randomly generated)
$archive \leftarrow \emptyset$

for each generations **do**
  Execution of each controller in simulation
  for each individual $i \in pop_{controller}$ **do**
    neigh(i) $\leftarrow$ The 15 individuals $\in (pop_{controller} \cup archive)$ similar to i
    Computation of the novelty objective: $Novelty(i) = \frac{\sum_{j \in neigh(i)} \|E_i - E_j\|}{|neigh(i)|}$
    Computation of the local competition objectives: $Local(i) = card(j \in neigh(i), perf(j) > perf(i))$
    if $Novelty(i) > \rho$ **then**
      Add the individual in the archive
    end if
    if $perf(i) >$ than the nearest individual in the archive **then**
      Swap individual i with the nearest individual in the archive
    end if
  end for
  Iteration of MOEA on $pop_{controller}$
end for

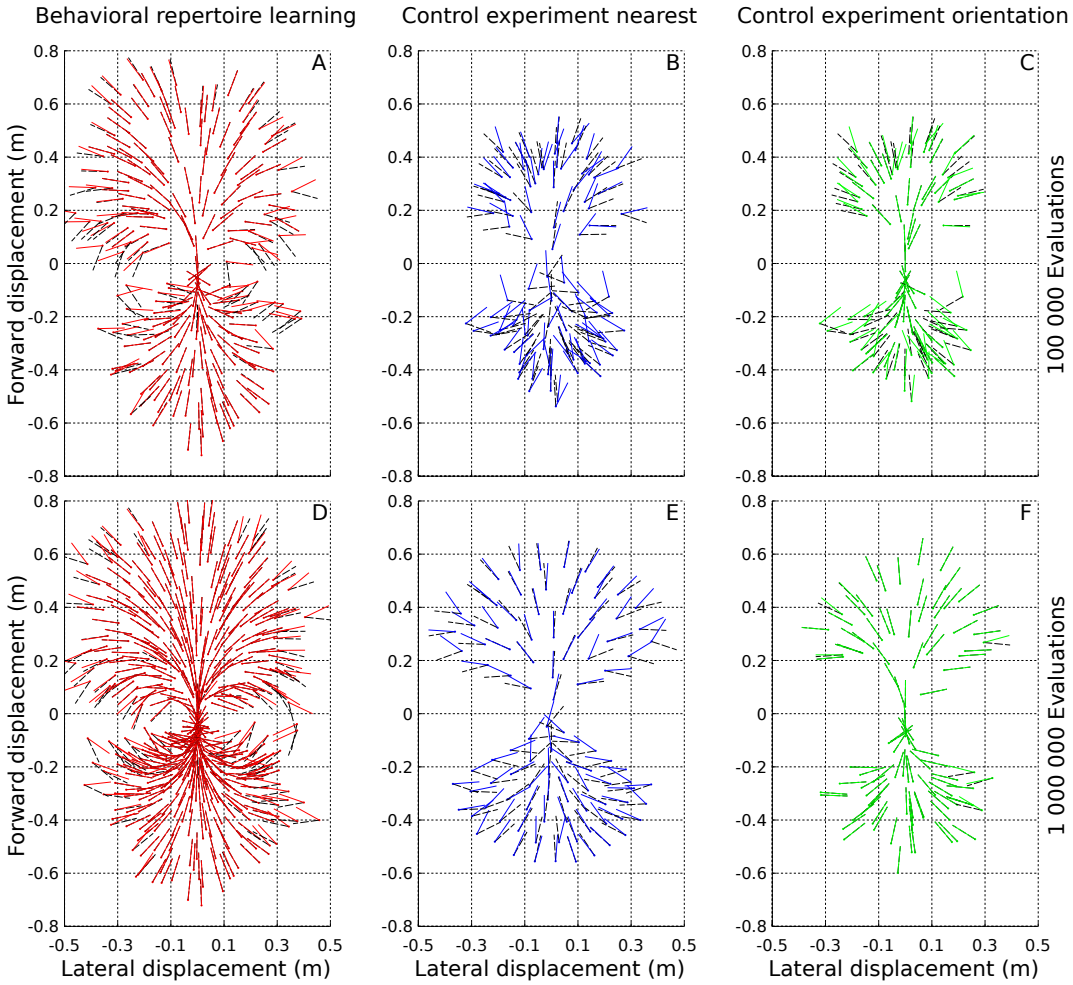

Figure 3: Typical results of the BR-Evolution (A,D) and the control experiments (B,E and C,F). A,B,C show the results after 100 000 evaluations, while D,E,F show those after 1 000 000 evaluations. Each dot corresponds to the endpoint of a controller. Colored solid lines represent the final orientation of the robot for each controller, while black dashed lines represent the desired orientation. The orientation error is the angle between solid and dashed lines.

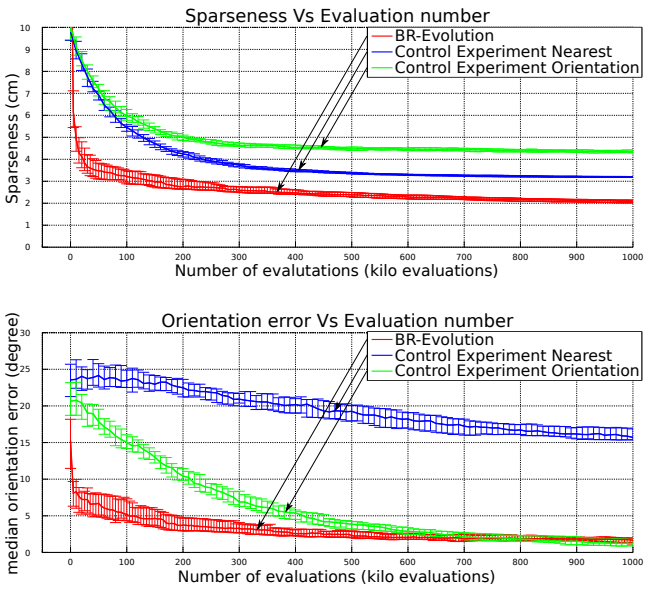

Figure 4: (Top) Variation of the sparseness of the controller collection. For each point of a one centimeter grid inside the ROI (figure 2), the distance from the nearest controller is computed. The sparseness value is the median of these distances. This graph plots the first three quartiles of the sparseness computed with 40 runs of the algorithms. (Bottom) Variation of the median of the orientation error over all the controllers inside the reachable area. This graph also plots the three first quartiles computed with 40 runs of the algorithms.

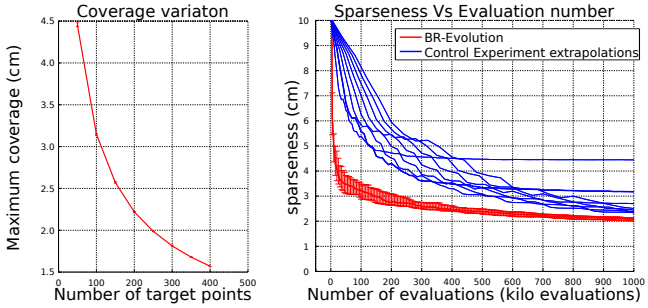

Figure 5: (Left) Sparseness ability of the control experiments according to the number of points. With more points, the sparseness value will be better (lower). (Right) Extrapolations of the variation of the sparseness for the "nearest" variant of the control experiment according to different number of targets. Each blue line is an extrapolation of the variation of the sparseness of the "nearest variant". They are based on a number of points starting from 50 to 400 with 50 points step. The variation of BR-Evolution is also plotted in red of comparison.

at the beginning of the "orientation" variant too, but, at the end, the error is negligible for the majority of controllers.

The BR-Evolution consistently leads to very small orientation errors (Figures 3 and 6); only few points have a significant error. We find these points in two regions, far from the starting point and directly on its side. These regions are characterized by their difficulty to be accessed, which stems from two main causes: the large distance to the starting point or the complexity of the required trajectory, which could be not feasible with the employed controller (Appendix B). For example the close lateral regions require executing a trajectory with a very high curvature, which can not be executed with the possible parameters of the controller. Moreover, the behaviors obtained in these regions are most of the time degenerated. Since accessing these points is difficult, finding better solutions is difficult for the evolutionary algorithm. We also observe a correlation between the density of controllers, the orientation error and the regions difficult to access (figure 6): The more a region is difficult to access, the less we find controllers and the less these controllers have a good orientation. For the others regions, the algorithm produces behaviors with various lengths and curvatures, covering all the reachable area of the robot.

In order to get a statistical point of view, we studied the median, over the 40 runs, of the sparseness of controllers inside the ROI (figure 4, Top). The BR-Evolution reaches low sparseness value with few evaluations. After 100 000 evaluations, it is able to generate behaviors covering the reachable space with an interval distance of about 3 cm. At the end of the process, the sparseness value is near 2 cm. With other experiments, the variation is slower and reaches a significantly higher level of sparseness (p-values $= 1.4 \times 10^{-14}$ with Wilcoxon rank-sum tests). The "orientation" variant of the control experiment presents the worst sparseness value ($> 4cm$). This result is expected because this variant favors behaviors with a good orientation even if they are far from their reference point. This phenomenon leads to a sample of the space less evenly distributed. The "nearest" variant achieves every target points, thus the sparseness value is better than with the "orientation" variant (3 cm vs 4cm, at the end of the experiment).

From the orientation point of view, our approach needs few evaluations to reach a low error value (< 5 degrees after 100 000 evaluations and < 1.7 degrees at the end). The variation of the "orientation" control experiment is slower and needs 750 000 evaluations to cross the curve of the BR-Evolution. At the end of the experiment this variant reaches a significantly lower error level (p-values $= 3.0 \times 10^{-7}$ with Wilcoxon rank-sum tests), but this corresponds to a difference of the medians of only 0.5 degrees. The "nearest" variant suffers from significantly higher orientation error (> 15 degrees, p-values $= 1.4 \times 10^{-14}$ with Wilcoxon rank-sum tests). This is expected because this variant selects behaviors taking into account only the distance from the target point. With this selection, the orientation aspect is neglected.

With the sets of reference points, we can compute the theoretical minimal sparseness value of the control experiments (figure 5, Left). For example, changing the number of targets from 100 to 200 will change the sparseness value from 3.14 cm to 2.22 cm. Nonetheless, doubling the number of points will double the required evaluations. Thanks to these values we can extrapolate the variation of the sparseness according to the number of points. For example, with the previously given values, we can predict that the graph

will reaches the 2.22 value and then scale the variation to fit this estimation. But we also know that this variation will be twice longer, then we also scale the temporal axis. Then we can trace the estimated variation of the sparseness value, for a given number of target points.

The extrapolations (figure 5, Right) show higher sparseness values compared to the BR-Evolution within the same execution time. Better values will be achieved with more evaluations. For instance, with 400 targets the sparseness value reaches 1.57 cm, but only after 4 millions of evaluations. This figure shows how our approach is faster than the control experiments regardless the number of reference points.

Figures 4 and 5 demonstrate how BR-Evolution is better both in the sparseness and in the orientation aspects compared than the proposed control experiments. Within few evaluations, reachable points are evenly distributed around the robot and corresponding behaviors are mainly well oriented.

An illustrating video is available on:
`http://youtu.be/2aTIL_c-qwA`

5. CONCLUSION AND DISCUSSION

To our knowledge, the BR-Evolution is the first method able to learn a large number of actions without testing each of them separately. With this technique, a large behavioral repertoire can be found with only one evolutionary process.

The BR-Evolution was evaluated on a simulated hexapod robot. With only 100 000 evaluations (1000 generations), it found a large controller collection that allows the robot to move through all its reachable space. The resulting controllers have high orientation accuracy, with an average error of 1.7 degrees. The BR-Evolution approach is at least 5 times faster and about 10 times more accurate than learning the controllers one by one. With an experiment focused on the orientation issue, our method reaches the same order of magnitude of accuracy but covers twice better the reachable area of the robot.

Overall, these experiments show that *the BR-Evolution is a powerful method for learning a large amount of accurate controllers and only within few generations.*

In future works, we plan to apply the algorithm to a real robot. We are aware of the difficulty in executing it directly on the robot, thus we project to use the transferability approach [15] to reduce the number of evaluations on the real robot. We also consider trying other types of actions and controllers, for example using pointing actions with a humanoid or central patterns generators.

Moreover, we want to investigate the ability of the BR-Evolution to autonomously explore and exploit the abilities of the agent. For instance, no indications are given to the algorithm about the ability of the robot to go backward or its inability to move more than 80 cm within 3 seconds. This attractive property of the BR-Evolution is similar to works in developmental robotics about "artificial curiosity"[1, 21]. It would be interesting to study the links between these approaches, which come from different branches of artificial intelligence.

6. ACKNOWLEDGMENTS

This work has been funded by the ANR Creadapt project (ANR-12-JS03-0009) and a DGA/UPMC scholarship for A.C..

7. REFERENCES

[1] A. Barto, S. Singh, and N. Chentanez. Intrinsically motivated learning of hierarchical collections of skills. *Proceedings of ICDL*, 2004.

[2] B. Biguer, M. Jeannerod, and C. Prablanc. The coordination of eye, head, and arm movements during reaching at a single visual target. *Experimental Brain Research*, 46(2):301–304, 1982.

[3] J. Bongard, V. Zykov, and H. Lipson. Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121, 2006.

[4] J. Clune, K. Stanley, R. Pennock, and C. Ofria. On the performance of indirect encoding across the continuum of regularity. *IEEE Transactions on Evolutionary Computation*, 15(3):346–367, 2011.

[5] K. Currie and A. Tate. O-plan: the open planning architecture. *Artificial Intelligence*, 52(1):49–86, 1991.

[6] T. Dean and M. Wellman. *Planning and control*. Morgan Kaufmann Publishers Inc., 1991.

[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[8] M. Graziano. The organization of behavioral repertoire in motor cortex. *Annual review of neuroscience*, 29(May):105–34, Jan. 2006.

[9] G. Hornby, S. Takamura, T. Yamamoto, and M. Fujita. Autonomous evolution of dynamic gaits with two quadruped robots. *IEEE Transactions on Robotics*, 21(3):402–410, 2005.

[10] R. Jacobs, M. Jordan, S. Nowlan, and G. Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[11] H. Kimura, T. Yamashita, and S. Kobayashi. Reinforcement learning of walking behavior for a four-legged robot. In *Proceedings of IEEE Conference on Decision and Control (CDC)*, volume 1, pages 411–416. IEEE, 2001.

[12] J. Knowles and D. Corne. Quantifying the effects of objective space dimension in evolutionary multiobjective optimization. In *Evolutionary Multi-Criterion Optimization*, pages 757–771. Springer, 2007.

[13] J. Kodjabachian and J.-A. Meyer. Evolution and development of neural controllers for locomotion, gradient-following, and obstacle-avoidance in artificial insects. *IEEE transactions on neural networks*, 9(5):796–812, Jan. 1998.

[14] N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proceedings of IEEE ICRA*, pages 2619–2624, 2004.

[15] S. Koos, J.-B. Mouret, and S. Doncieux. The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*, pages 122–145, 2013.

[16] J. Lehman and K. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

[17] J. Lehman and K. Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of GECCO*, pages 211–218. ACM, 2011.

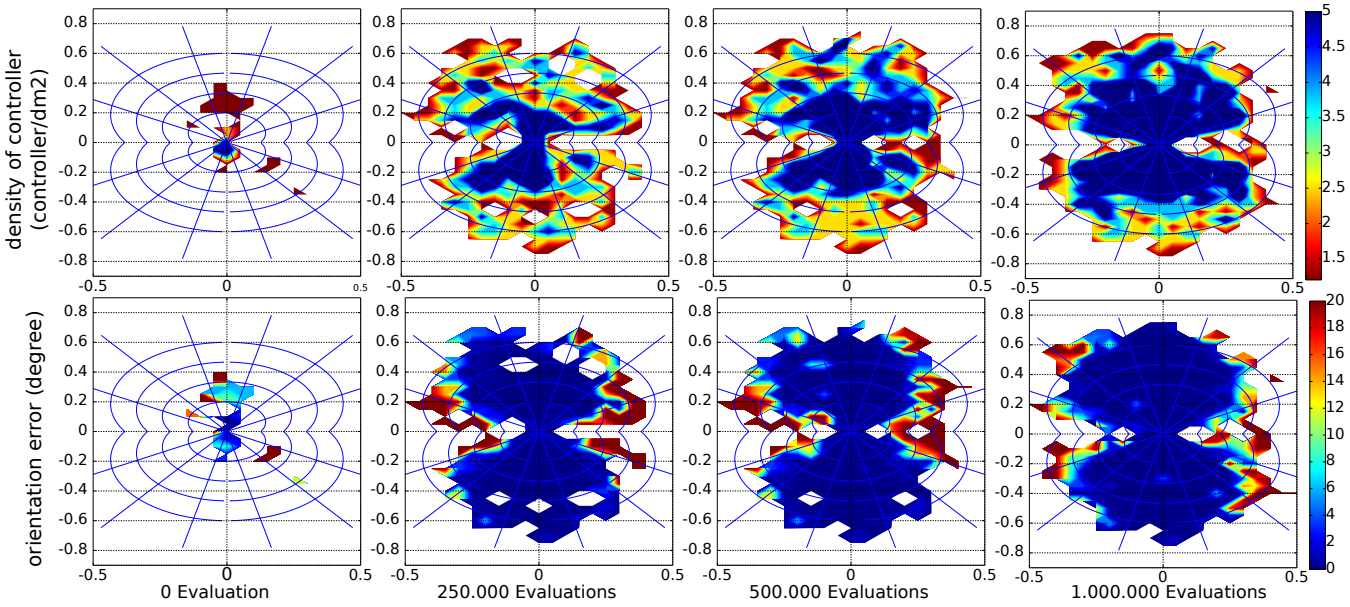

Figure 6: (Top) Variation of density of controller (number of controller per dm^2). (Bottom) Variation of the orientation error (given by the nearest controller) along a typical run.

[18] S. Mahdavi and P. Bentley. Innately adaptive robotics through embodied evolution. *Autonomous Robots*, 20(2):149–163, 2006.

[19] J.-B. Mouret and S. Doncieux. Sferes$_{v2}$: Evolvin' in the Multi-Core World. In *Proceedings of IEEE CEC*, pages 4079–4086, 2010.

[20] J.-B. Mouret, S. Doncieux, and J.-A. Meyer. Incremental evolution of target-following neuro-controllers for flapping-wing animats. *From Animals to Animats 9*, 2006.

[21] P.-Y. Oudeyer. Intelligent adaptive curiosity: a source of self-development. *Proceedings of the Fourth International Workshop on Epigenetic Robotics*, 117:127–130, 2004.

[22] S. Russell, P. Norvig, and E. Davis. *Artificial intelligence: a modern approach*. Prentice hall Upper Saddle River, NJ, 2010.

[23] G. Seber. *Multivariate observations*, volume 41. Wiley New York, 1984.

[24] R. Tedrake, T. Zhang, and H. Seung. Learning to walk in 20 minutes. In *Proceedings of Yale workshop on Adaptive and Learning Systems*, 2005.

APPENDIX

A. ALGORITHM PARAMETERS

The source-code of our experiments can be downloaded on: `http://pages.isir.upmc.fr/evorob_db/moin.wsgi/BehavioralRepertoireLearningInRobotics`

BR-Evolution experiments:

- Population size: 100 individuals
- Number of generation: 10 000
- Mutation rate: 10% on each parameters
- Crossover: disabled
- ρ: 0.10 m
- ρ variation: none

Control experiments:

- Population size : 100 individuals
- Number of generation : 50 0000 (100 * 500)
- Mutation rate : 10% on each parameters
- Crossover : disable

B. PARAMETRIZED CONTROLLER

The simulated robot is a hexapod with 18 Degrees of Freedom (DOF), 3 for each leg (Fig. 1). The first servo controls the horizontal orientation of the leg and the two others control its elevation. The kinematic scheme of the robot is pictured on Figure 1(Right). The movement of each DOF is governed by a periodic function that computes its angular position as a function γ of time t, amplitude α and phase ϕ:

$$\gamma(t,\ \alpha,\ \phi) = \alpha \cdot \tanh\left(4 \cdot \sin\left(2 \cdot \pi \cdot (t + \phi)\right)\right) \qquad (3)$$

where α and ϕ are the parameters that define the amplitude of the movement and the phase shift of γ, respectively. The frequency is fixed. Angular positions are sent to the servos every 30 ms. The main feature of this particular function is that, thanks to the tanh function, the control signal is constant during a large part of each cycle, thus allowing the robot to stabilize itself. In order to keep the "tibia" of each leg vertical, the same control signal is used for the two last servos. Consequently, positions sent to the i^{th} servos are:

- $\gamma(t,\ \alpha_1^i,\ \phi_1^i)$ for DOF 1;
- $\gamma(t,\ \alpha_2^i,\ \phi_2^i)$ for DOFs 2 and 3.

There are 4 parameters for each leg (α_1^i, α_2^i, ϕ_1^i, ϕ_2^i), therefore each controller is fully described by 24 parameters. Five values are available for each paramater $(0, 0.25, 0.5, 0.75, 1)$. By varying these 24 parameters, numerous gaits are possible, from purely quadruped gaits to classic tripod gaits in every directions.