
Evolving a Behavioral Repertoire for a Walking Robot

A. Cully

cully@isir.upmc.fr

Sorbonne Universités, UPMC Univ Paris 06, UMR 7222, ISIR, F-75005, Paris, France
CNRS, UMR 7222, ISIR, F-75005, Paris, France

J.-B. Mouret

mouret@isir.upmc.fr

Sorbonne Universités, UPMC Univ Paris 06, UMR 7222, ISIR, F-75005, Paris, France
CNRS, UMR 7222, ISIR, F-75005, Paris, France

Abstract

Numerous algorithms have been proposed to allow legged robots to learn to walk. However, the vast majority of these algorithms is devised to learn to walk in a straight line, which is not sufficient to accomplish any real-world mission. Here we introduce the Transferability-based Behavioral Repertoire Evolution algorithm (TBR-Evolution), a novel evolutionary algorithm that simultaneously discovers several hundreds of simple walking controllers, one for each possible direction. By taking advantage of solutions that are usually discarded by evolutionary processes, TBR-Evolution is substantially faster than independently evolving each controller. Our technique relies on two methods: (1) novelty search with local competition, which searches for both high-performing and diverse solutions, and (2) the transferability approach, which combines simulations and real tests to evolve controllers for a physical robot. We evaluate this new technique on a hexapod robot. Results show that with only a few dozen short experiments performed on the robot, the algorithm learns a repertoire of controllers that allows the robot to reach every point in its reachable space. Overall, TBR-Evolution opens a new kind of learning algorithm that simultaneously optimizes all the achievable behaviors of a robot.

Keywords

Evolutionary Algorithms, Evolutionary Robotics, Mobile Robotics, Behavioral Repertoire, Exploration, Novelty Search, Hexapod Robot.

1 Introduction

Evolving gaits for legged robots has been an important topic in evolutionary computation for the last 25 years (de Garis, 1990; Lewis et al., 1992; Kodjabachian and Meyer, 1998; Hornby et al., 2005; Clune et al., 2011; Yosinski et al., 2011; Samuelsen and Glette, 2014). That legged robots is a classic of evolutionary robotics is not surprising (Bongard, 2013): legged locomotion is a difficult challenge in robotics that evolution by natural selection solved in nature; evolution-inspired algorithm may do the same for artificial systems. As argued in many papers, evolutionary computation could bring many benefits to legged robotics, from making it easier to design walking controllers (e.g., Hornby et al. (2005)), to autonomous damage recovery (e.g., Bongard et al. (2006); Koos et al. (2013a)). In addition, in an embodied cognition perspective (Wilson, 2002; Pfeifer and Bongard, 2007; Pfeifer et al., 2007), locomotion is one of the most fundamental skills of animals, and therefore it is one of the first skill needed for embodied agents.

It could seem more confusing that evolutionary computation has failed to be central in legged robotics, in spite of the efforts of the evolutionary robotics community (Raibert, 1986; Siciliano and Khatib, 2008). In our opinion, this failure stems from at least two reasons: (1) most evolved controllers are almost useless in robotics because they are limited to walking in a straight line at constant speed (e.g. Hornby et al. (2005); Bongard et al. (2006); Koos et al. (2013a)), whereas a robot that only walks in a straight line is obviously unable to accomplish any mission; (2) evolutionary algorithms typically require evaluating the fitness function thousands of times, which is very hard to achieve with a physical robot. The present article introduces TBR-Evolution (Transferability-based Behavioral Repertoire Evolution), a new algorithm that addresses these two issues at once.

Evolving controllers to make a robot walk in any direction can be seen as a generalization of the evolution of controllers for forward walking. A straightforward idea is to add an additional input to the controller that describes the target direction, then evolve controllers that use this input to steer the robot (Mouret et al., 2006). Unfortunately, this approach requires testing each controller for several directions in the fitness function, which substantially increases the time required to find a controller. In addition, an integrated controller that can use a direction input is likely to be more difficult to find than a controller that can only do forward walking.

An alternate idea is to see walking in every direction as a problem of learning how to do many different – but related – tasks. In this case, an evolutionary algorithm could search for a *repertoire of simple controllers* that would contain a different controller for each possible direction. This method circumvents the challenge of learning a complex controller and can be combined with high level algorithms (e.g. planning algorithms) that successively select controllers to drive the robot. Nevertheless, evolving a controller repertoire typically involves as many evolutionary processes as there are target points in the repertoire. Evolution is consequently slowed down by a factor equal to the number of targets. With existing evolution methods, repertoires of controllers are in effect limited to a few targets, because 20 minutes (Koos et al., 2013a) to dozens of hours (Hornby et al., 2005) are needed to learn how to reach a single target.

Our algorithm aims to find such a repertoire of simple controllers, but *in a single run*. It is based on a simple observation: with a classic evolutionary algorithm, when a robot learns to reach a specific target, the learning process explores many different potential solutions, with many different outcomes. Most of these potential solutions are discarded because they are deemed poorly-performing. Nevertheless, while being useless for the considered objective, these inefficient behaviors can be useful for other objectives. For example, a robot learning to walk in a straight line usually encounters many turning gaits during the search process, before converging towards straight line locomotion.

To exploit this idea, TBR-Evolution takes inspiration from the “Novelty Search” algorithm (Lehman and Stanley, 2011a), and in particular its variant the “Novelty Search with Local Competition” (Lehman and Stanley, 2011b). Instead of rewarding candidate solutions that are the closest to the objective, this recently introduced algorithm explicitly searches for behaviors that are different from those previously seen. The local competition variant adds the notion of a quality criterion which is optimized within each individual’s niche. As shown in the rest of the present article, searching for many different behaviors during a single execution of the algorithm allows the evolutionary process to efficiently create a repertoire of high-performing walking gaits.

To further reduce the time required to obtain a behavioral repertoire for the robot,

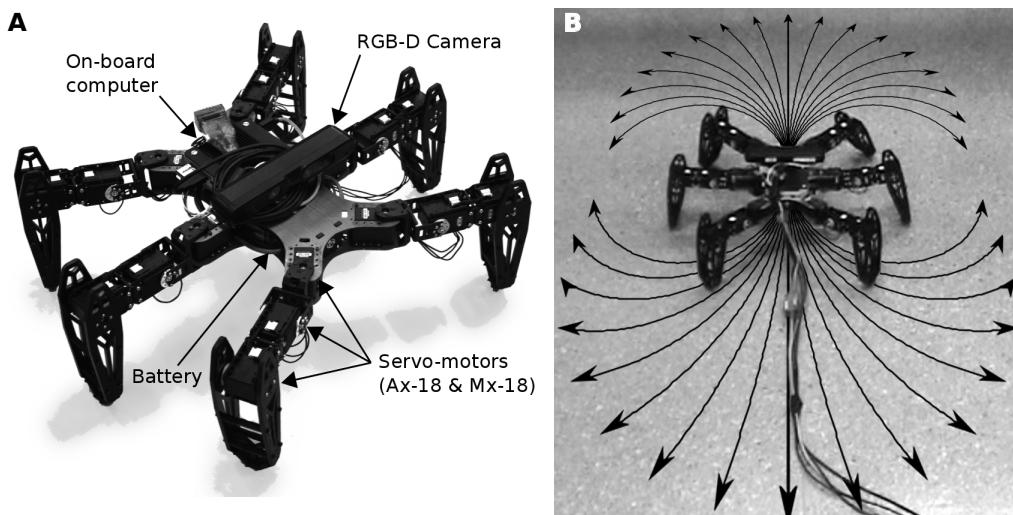


Figure 1: (Right) The hexapod robot. It has 18 degrees of freedom (DOF), 3 for each leg. Each DOF is actuated by position-controlled servos (Dynamixel actuators). A RGB-D camera (Asus Xtion) is screwed on the top of the robot. The camera is used to estimate the forward displacement of the robot thanks to a RGB-D Simultaneous Localization And Mapping (SLAM) algorithm (Endres et al., 2012) from the ROS framework (Quigley et al., 2009). (Left) Goal of TBR-Learning. Our algorithm allows the hexapod robot to learn to walk in every direction with a single run of the evolutionary algorithm.

TBR-Evolution relies on the *transferability approach* (Koos et al., 2013b; Mouret et al., 2012), which combines simulations and tests on the physical robot to find solutions that perform similarly in simulation and in reality. The advantages of the transferability approach is that evolution occurs in simulation but the evolutionary process is driven towards solutions that are likely to work on the physical robot. In recent experiments, this approach led to the successful evolution of walking controllers for quadruped (Koos et al., 2013b), hexapod (Koos et al., 2013a), and biped robots (Oliveira et al., 2013), with no more than 25 tests on the physical robot.

We evaluate our algorithm on two sets of experiments. The first set aims to show that learning simultaneously all the behaviors of a repertoire is faster than learning each of them separately¹. We chose to perform these experiments in simulation to gather extensive statistics. The second set of experiments evaluates our method on a physical hexapod robot (Fig. 1, left) that has to walk forward, backward, and turn in both directions, all at different speeds (Fig. 1, right). We compare our results to learning independently each controller. All our experiments utilize embedded measurements to evaluate the fitness, an aspect of autonomy only considered in a handful of gait discovery experiments (Kimura et al., 2001; Hornby et al., 2005).

2 Background

2.1 Evolving Walking Controllers

We call *Walking Controller* the software module that rhythmically drives the motors of the legged robot. We distinguish two categories of controllers: *un-driven controllers* and *inputs-driven controllers*. An un-driven controller always executes the same gait, while an inputs-driven controller can change the robot's movements according to an input (e.g. a speed or a direction reference). Inputs-driven controllers are typically combined with decision or planning algorithms (Russell et al., 2010; Currie and Tate, 1991; Dean and Wellman, 1991; Kuffner and LaValle, 2000) to steer the robot. These two categories contain, without distinctions, both open-loop and closed-loop controllers and can be designed using various controller and genotype structures. For example, walking gait evolution or learning has been achieved on legged robots using parametrized periodic functions (Koos et al., 2013a; Chernova and Veloso, 2004; Hornby et al., 2005; Tarapore and Mouret, 2014a,b), artificial neural networks with both direct or generative encoding (Clune et al., 2011; Valsalam and Miikkulainen, 2008; Tarapore and Mouret, 2014a,b), Central Pattern Generators (Kohl and Stone, 2004; Ijspeert et al., 2007), or graph-based genetic programming (Filliat et al., 1999; Gruau, 1994).

When dealing with *physical* legged robots, the majority of studies only considers the evolution of un-driven walking controllers and, most of the time, the task consists in finding a controller that maximizes the forward walking speed (Zykov et al., 2004; Chernova and Veloso, 2004; Hornby et al., 2005; Berenson et al., 2005; Yosinski et al., 2011; Mahdavi and Bentley, 2006). Papers on alternatives to evolutionary algorithms, like policy gradients (Kohl and Stone, 2004; Tedrake et al., 2005) or Bayesian optimization (Calandra et al., 2014; Lizotte et al., 2007), are also focused on robot locomotion along a straight line.

Comparatively few articles deal with controllers able to turn or to change the walking speed according to an input, especially with a physical robot. Inputs-driven controllers usually need to be tested on each possible input during the learning process or

¹This experiment is partly based on the preliminary results published in a conference paper (Cully and Mouret, 2013)

to be learned with an incremental process, which significantly increases both the learning time and the difficulty compared to learning an un-driven controller. Filliat et al. (1999) proposed such a method, that evolves a neural network to control a hexapod robot. Their neural network is learned with several steps: first, the network is learned in order to walk in a straight line; in a second step, a second neural network is evolved on top of the walking controller be able to execute turning manoeuvres. In a related task (flapping wing flight), Mouret et al. (2006) proposed another approach, where an evolutionary algorithm is used to design a neural network that pilots a simulated flapping robot; the network was evaluated by its ability to drive the robot to 8 different targets and the reward function was the sum of the distances to the targets.

Overall, many methods exist to evolve un-driven controllers, while methods for learning inputs-driven controllers are very time-expensive, difficult to apply on a physical robot, and require an extensive amount of expert knowledge. To our knowledge, no current technique is able to make a physical robot learning to walk in multiple directions in less than a dozen hours. In this paper, we sidestep many of the challenges raised by input-driven controllers while being able to drive a robot in every direction: we propose to abandon input-driven controllers, and, instead, search for a large number of simple, un-driven controllers, one for each possible direction.

2.2 Transferability approach

Most of the previously described methods are based on stochastic optimization algorithms that need to test a high number of candidate solutions. Typically, several thousands of tests are performed with policy gradient methods (Kohl and Stone, 2004) and hundreds of thousands with evolutionary algorithms (Clune et al., 2011). This high number of tests is a major problem when they are performed on a physical robot. An alternative is to perform the learning process in simulation and then apply the result to the robot. Nevertheless, solutions obtained in simulation often do not work well on the real device, because simulation and reality never match perfectly. This phenomenon is called *the Reality Gap* (Jakobi et al., 1995; Koos et al., 2013b).

The transferability approach (Koos et al., 2013b,a; Mouret et al., 2012) crosses this gap by finding behaviors that act similarly in simulation and in reality. During the evolutionary process, a few candidate controllers are transferred to the physical robot to measure the behavioral differences between the simulation and the reality; these differences represent the *transferability value* of the solutions. With these few transfers, a *regression model* is built up to map solution descriptors to an estimated transferability value. The regression model is then used to predict the transferability value of untested solutions. The transferability approach uses a multi-objective optimization algorithm to find solutions that maximize both task-efficiency (e.g. forward speed, stability) and the estimated transferability value.

This mechanism drives the optimization algorithm towards solutions that are both efficient in simulation and transferable (i.e. that act similarly in the simulation and in the reality). It allows the algorithm to exploit the simulation and consequently to reduce the number of tests performed on the physical robot.

This approach was successfully used with an E-puck robot in a T-maze and with a quadruped robot that evolved to walk in a straight line with a minimum of transfers on the physical robots (Koos et al., 2013b). The reality gap phenomenon was particularly apparent in the quadruped experiment: with a controller optimized only in simulation, the virtual robot moved 1.29m (in 10s) but when the same controller was applied on the physical robot, it only moved 0.41m. With the transferability approach, the obtained

solution walked $1.19m$ in simulation and $1.09m$ in reality. These results were found with only 11 tests on the physical robot and 200,000 evaluations in simulation. This approach has also been applied for humanoid locomotion (Oliveira et al., 2013) and damage recovery on a hexapod robot (Koos et al., 2013a).

Since the transferability approach is one of the most practical tool to apply stochastic optimization algorithms on physical robots, it constitutes an element of our method.

2.3 Novelty Search with Local Competition

A longstanding challenge in artificial life is to craft an algorithm able to discover a wide diversity of interesting artificial creatures. While evolutionary algorithms are good candidates, they usually converge to a single species of creatures. In order to overcome this issue, Lehman and Stanley recently proposed a method called *Novelty search with local competition* (Lehman and Stanley, 2011b). This method, based on multi-objective evolutionary algorithms, combines the exploration abilities of the Novelty Search algorithm (Lehman and Stanley, 2011a) with a performance competition between similar individuals.

The Novelty Search with Local Competition simultaneously optimizes two objectives for an individual c : (1) the novelty objective ($\text{novelty}(c)$), which measures how novel is the individual compared to previously encountered ones, and (2) the local competition objective ($Q\text{rank}(c)$), which compares the individual's quality ($\text{quality}(c)$) to the performance of individuals in a neighborhood, defined with a morphological distance.

With these two objectives, the algorithm favors individuals that are new, those that are more efficient than their neighbors and those that are optimal trade-offs between novelty and "local quality". Both objectives are evaluated thanks to an *archive*, which records all encountered family of individuals and allows the algorithm to define neighborhoods for each individual. The novelty objective is computed as the average distance between the current individual and its neighbors, and the local competition objective is the number of neighbors that c outperforms according to the quality criterion $\text{quality}(i)$.

The authors successfully applied this method to generate a high number of creatures with different morphologies, all able to walk in a straight line. The algorithm found a heterogeneous population of different creatures, from little hoppers to imposing quadrupeds, all walking at different speeds according to their stature. We will show in this paper how this algorithm can be modified to allow a single robot to achieve several different actions (i.e. directions of locomotion).

3 TBR-Evolution

3.1 Main ideas

Some complex problems are easier to solve when they are split into several sub-problems. Thus, instead of using a single and complex solution, it is relevant to search for several simple solutions that solve a part of the problem. This principle is often successfully applied in machine learning: mixtures of experts (Jacobs et al., 1991) or boosting (Schapire, 1990) methods train several weak classifiers on different sub-parts of a problem. Performances of the resulting set of classifiers are better than those of a single classifier trained on the whole problem.

The TBR-Evolution algorithm enables the application of this principle to robotics and, particularly, to legged robots that learn to walk. Instead of learning a complex, inputs-driven controller that generates gaits for every direction, we consider a reper-

toire of un-driven controllers, where each controller is able to reach a different point of the space around the robot. This repertoire gathers a high number of efficient and easy-to-learn controllers.

Because of the required time, independently learning dozens of controllers is prohibitively expensive, especially with a physical robot. To avoid this issue, the TBR-Evolution algorithm transforms the problem of learning a repertoire of controllers into a problem of evolving a heterogeneous population of controllers. Thus the problem can be solved with an algorithm derived from novelty search with local competition (Lehman and Stanley, 2011b): instead of generating virtual creatures with various morphologies that execute the same action, TBR-Evolution generates a repertoire of controller, each executing a different action, working on the same creature. By simultaneously learning all the controllers without the discrimination of a specified goal, the algorithm recycles interesting controllers, which are typically wasted with classical learning methods. This enhances its optimizing abilities compared to classic optimization methods.

Furthermore, our algorithm incorporates the transferability approach (Koos et al., 2013b) to reduce the number of tests on the physical robot during the evolutionary process. The transferability approach and novelty search with local competition can be combined because they are both based on multi-objective optimization algorithms. By combining these two approaches, the behavioral repertoire is generated in simulation with a virtual robot and only a few controller executions are performed on the physical robot. These trials guide the evolutionary process to solutions that work similarly in simulation and in reality (Koos et al., 2013a,b).

The minimization of the number of tests on the physical robot and the simultaneous evolution of many controllers are the two assets that allow the TBR-Evolution algorithm to require significantly less time than classical methods.

More technically, the TBR-Evolution algorithm relies on four principles, detailed in the next sections:

- a stochastic, black box, multi-objective optimization algorithm simultaneously optimizes 3 objectives, all evaluated in simulation: (1) the novelty of the gait, (2) the local rank of quality and (3) the local rank of estimated transferability:

$$\text{maximize} \left\{ \begin{array}{l} \text{Novelty}(c) \\ -\text{Qrank}(c) \\ -\widehat{\text{Trank}}(c) \end{array} \right.$$

- the transferability function is periodically updated with a test on the physical robot;
- when a controller is novel enough, it is saved in the *novelty archive*;
- when a controller has a better quality than the one in the archive that reaches the same endpoint, it substitutes the one in the archive.

Algorithm 1 describes the whole algorithm in pseudo-code.

3.2 Objectives

The novelty objective fosters the exploration of the reachable space. A controller is deemed as novel when the controlled individual reaches a region where none, or few of the previously encountered gaits were able to go (starting from the same point). The

novelty score of a controller \mathbf{c} ($Novelty(\mathbf{c})$) is set as the average distance between the endpoint of the current controller (\mathcal{E}_c) and the endpoints of controllers contained in its neighborhood ($\mathcal{N}(\mathbf{c})$):

$$Novelty(\mathbf{c}) = \frac{\sum_{\mathbf{j} \in \mathcal{N}(\mathbf{c})} \|\mathcal{E}_{simu}(\mathbf{c}) - \mathcal{E}_{simu}(\mathbf{j})\|}{card(\mathcal{N}(\mathbf{c}))} \quad (1)$$

To get high novelty scores, individuals have to follow trajectories leading to endpoints far from the rest of the population. The population will thus explore all the area reachable by the robot. Each time a controller with a novelty score exceeds a threshold (ρ), this controller is saved in an *archive*. Given this archive and the current population of candidate solutions, a neighborhood is defined for each controller ($\mathcal{N}(\mathbf{c})$). This neighborhood regroups the k controllers that arrive closest to the controller \mathbf{c} (the parameters' values are detailed in the appendix).

The local quality rank promotes controllers that show particular properties, like stability or accuracy. These properties are evaluated by the quality score ($quality(\mathbf{c})$), which depends on implementation choices and particularly on the type of controllers used (we will detail its implementation in section 4.1). In other words, among several controllers that reach the same point, the quality score defines which one should be promoted. For a controller \mathbf{c} , the rank ($Qrank(\mathbf{c})$) is defined as the number of controllers from its neighborhood that outperform its quality score: minimizing this objective allows the algorithm to find controllers with better quality than their neighbors.

$$Qrank(\mathbf{c}) = card(\mathbf{j} \in \mathcal{N}(\mathbf{c}), quality(\mathbf{c}) < quality(\mathbf{j})) \quad (2)$$

The local transferability rank ($\widehat{Trank}(\mathbf{c})$, equation 3) works as a second local competition objective, where the estimation of the transferability score ($\hat{T}(\text{des}(\mathbf{c}))$) replaces the quality score. Like in (Koos et al., 2013b), this estimation is obtained by periodically repeating three steps: (1) a controller is randomly selected in the current population or in the archive and then downloaded and executed on the physical robot, (2) the displacement of the robot is estimated thanks to an embedded sensor, and (3) the distance between the endpoint reached in reality and the one in simulation is used to feed a regression model (\hat{T} , here a support vector machine (Chang and Lin, 2011)). This distance defines the transferability score of the controller. This model maps a behavioral descriptor of a controller ($\text{des}(\mathbf{c})$), which is obtained in simulation, with an approximation of the transferability score.

Thanks to this descriptor, the regression model predicts the transferability score of each controller in the population and in the archive.

$$\widehat{Trank}(\mathbf{c}) = card(\mathbf{j} \in \mathcal{N}(\mathbf{c}), \hat{T}(\text{des}(\mathbf{c})) < \hat{T}(\text{des}(\mathbf{j}))) \quad (3)$$

3.3 Archive management

In the original novelty search with local competition (Lehman and Stanley, 2011b), the archive aims at recording all encountered solutions, but only the first individuals that have a new morphology are added to the archive. The next individuals with the same morphology are not saved, even if they have better performances. In the TBR-Evolution algorithm, the novelty archive represents the resulting repertoire of controllers, and thus has to gather only the best controllers for each region of the reachable space.

For this purpose, the archive is differently managed than in the novelty search: during the learning process, if a controller of the population has better scores

($quality(\mathbf{c})$ or $\hat{\mathcal{T}}(\mathbf{c})$) than the closest controller in the archive, the one in the archive is replaced by the better one. These comparisons are made with a priority among the scores to prevent circular permutations. If the transferability score is lower than a threshold (τ), only the transferability scores are compared, otherwise we compare the quality scores. This mechanism allows the algorithm to focus the search on transferable controllers instead of searching efficient, but not transferable, solutions. Such a priority is important, as the performances of non-transferable controllers may not be reproducible on the physical robot.

Algorithm 1 TBR-Evolution algorithm (G generations, T transfers' period)

```

procedure TBR-EVOLUTION
     $pop \leftarrow \{c^1, c^2, \dots, c^S\}$  (randomly generated)
     $archive \leftarrow \emptyset$ 
    for  $g = 1 \rightarrow G$  do
        for all controller  $\mathbf{c} \in pop$  do
            Execution of  $\mathbf{c}$  in simulation
        if  $g \equiv 0[T]$  then
            TRANSFERABILITY UPDATE( $\mathbf{c}$ )
        for all controller  $\mathbf{c} \in pop$  do
            OBJECTIVE UPDATE( $\mathbf{c}$ )
            ARCHIVE MANAGEMENT( $\mathbf{c}$ )
        Iteration of NSGA-II on  $pop$ 
    return  $archive$ 

procedure TRANSFERABILITY UPDATE( $\mathbf{c}$ )
    Random selection of  $\mathbf{c}^* \in pop \cup archive$  and transfer on the robot
    Estimation of the endpoint  $\mathcal{E}_{real}(\mathbf{c}^*)$ 
    Estimation of the exact transferability value  $|\mathcal{E}_{simu}(\mathbf{c}^*) - \mathcal{E}_{real}(\mathbf{c}^*)|$ 
    Update of the approximated transferability function  $\hat{\mathcal{T}}$ 

procedure OBJECTIVES UPDATE( $\mathbf{c}$ )
     $\mathcal{N}(\mathbf{c}) \leftarrow$  The 15 controllers( $\in pop \cup archive$ ) closest to  $\mathcal{E}_{simu}(\mathbf{c})$ 
    Computation of the novelty objective:
         $Novelty(\mathbf{c}) = \frac{\sum_{\mathbf{j} \in \mathcal{N}(\mathbf{c})} \|\mathcal{E}_{simu}(\mathbf{c}) - \mathcal{E}_{simu}(\mathbf{j})\|}{|\mathcal{N}(\mathbf{c})|}$ 
    Computation of the local rank objectives:
         $Qrank(\mathbf{c}) = |\mathbf{j} \in \mathcal{N}(\mathbf{c}), quality(\mathbf{c}) < quality(\mathbf{j})|$ 
         $\widehat{Trank}(\mathbf{c}) = |\mathbf{j} \in \mathcal{N}(\mathbf{c}), \hat{\mathcal{T}}(\mathbf{des}(\mathbf{c})) < \hat{\mathcal{T}}(\mathbf{des}(\mathbf{j}))|$ 

procedure ARCHIVE MANAGEMENT( $\mathbf{c}$ )
    if  $Novelty(\mathbf{c}) > \rho$  then
        Add the individual to  $archive$ 
     $\mathbf{c}_{nearest} \leftarrow$  The controller  $\in archive$  nearest to  $\mathcal{E}_{simu}(\mathbf{c})$ 
    if  $\hat{\mathcal{T}}(\mathbf{des}(\mathbf{c})) > \tau$  and  $quality(\mathbf{c}) > quality(\mathbf{c}_{nearest})$  then
        Replace  $\mathbf{c}_{nearest}$  by  $\mathbf{c}$  in the archive
    else if  $\hat{\mathcal{T}}(\mathbf{des}(\mathbf{c})) > \hat{\mathcal{T}}(\mathbf{des}(\mathbf{c}_{nearest}))$  then
        Replace  $\mathbf{c}_{nearest}$  by  $\mathbf{c}$  in the archive

```

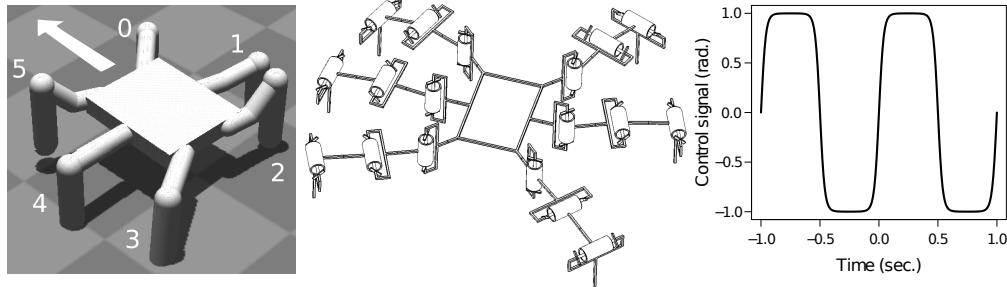


Figure 2: (Left) Snapshot of the simulated robot in our ODE-based physics simulator. (Center) Kinematic scheme of the robot. The cylinders represent actuated pivot joints. (Right) Control function $\gamma(t, \alpha, \phi)$ with $\alpha = 1$ and $\phi = 0$.

4 Experimental Validation

We evaluate the TBR-Evolution algorithm on two different experiments, which both consist in evolving a repertoire of controllers to access to the whole vicinity of the robot. In the first experiment, the algorithm is applied on a simulated robot (Fig. 2, left), consequently the transferability aspect of the algorithm is disabled. The goal of this experiment is to show the benefits of evolving simultaneously all the behaviors of the repertoire instead of evolving them separately. The second experiment applies the algorithm directly on a physical robot (Fig 1, left). For this experiment, the transferability aspect of the algorithm is enabled and the experiment shows how the behavioral repertoire can be learned with a few trials on the physical robot.

4.1 Implementation choices

The pseudo-code of the algorithm is presented in Algorithm 1. The TBR-Evolution algorithm uses the same variant of NSGA-II (Deb et al., 2002) as the novelty search with local competition (Lehman and Stanley, 2011b). The simulation of the robot is based on the *Open Dynamic Engine* (ODE) and the transferability function \hat{T} uses the ν -Support Vector Regression algorithm with linear kernels implemented in the library *libsvm* (Chang and Lin, 2011) (learning parameters set to default values). All the algorithms are implemented in the Sferes_{v2} framework (Mouret and Doncieux, 2010) (parameters and source code are detailed in appendix). The simulated parts of the algorithms are computed on a cluster of 5 quad-core Xeon-E5520@2.27GHz computers.

4.1.1 Robot

Both the virtual and the physical robots have the same kinematic scheme (see figure 2 center). They have 18 degrees of freedom, 3 per leg. The first joint of each leg controls the direction of the leg while the two others define its elevation and extension. The virtual robot is designed to be a “virtual copy” of the physical hexapod: it has the same mass for each of its body parts, and the physical simulator reproduces the dynamical characteristics of the servos. On the physical robot, the estimations of the covered distance are acquired with a Simultaneous Localisation and Mapping (SLAM) algorithm based on the embedded RGB-D sensor (Endres et al., 2012).

4.1.2 Genotype and Controller

The same genotype and controller structure are used for the two sets of experiments. The genotype is a set of 24 parameter values defining the angular position of each leg joint with a periodic function γ of time t , parametrized by an amplitude α and a phase shift ϕ (Fig. 2, right):

$$\gamma(t, \alpha, \phi) = \alpha \cdot \tanh(4 \cdot \sin(2 \cdot \pi \cdot (t + \phi))) \quad (4)$$

Angular positions are updated and sent to the servos every 30ms. The main feature of this particular function is that the control signal is constant during a large portion of each cycle, thus allowing the robot to stabilize itself. In order to keep the “tibia” of each leg vertical, the control signal of the third servo is the opposite of the second one. Consequently, positions sent to the i^{th} leg are:

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

The 24 parameters can each have five different values (0, 0.25, 0.5, 0.75, 1) and with their variations, numerous gaits are possible, from purely quadruped gaits to classic tripod gaits.

For the genotype mutation, each parameter value has a 10% chance of being changed to any value in the set of possible values, with the new value chosen randomly from a uniform distribution over the possible values. For both of the experiments, the crossover is disabled.

Compared to classic evolutionary algorithms, TBR-Evolution only changes the way individuals are selected. As a result, it does not put any constraint on the type of controllers, and many other controllers are conceivable (e.g. bio-inspired central pattern generators (Sproewitz et al., 2008; Ijspeert, 2008), dynamic movement primitives (Schaal, 2003) or evolved neural networks (Yosinski et al., 2011; Clune et al., 2011)).

4.1.3 Endpoints of a controller

The endpoint of a controller (in simulation or in reality) is the position of the center of the robot’s body projected in the horizontal plane after running the controller for 3 seconds:

$$\mathcal{E}(\mathbf{c}) = \left\{ \begin{array}{l} \text{center}_x(t=3s) - \text{center}_x(t=0s) \\ \text{center}_y(t=3s) - \text{center}_y(t=0s) \end{array} \right\}$$

4.1.4 Quality Score

To be able to sequentially execute saved behaviors, special attention is paid to the final orientation of the robot. Because 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. To facilitate chaining controllers, we encourage behaviors that end their movements with an orientation aligned with their trajectory.

The robot cannot execute arbitrary trajectories with a single controller because controllers are made of simple periodic functions. For example, it cannot begin its movement by a turn and then go straight. With this controller, the robot can only follow trajectories with a constant curvature, but it can still move sideways, or even turn around itself while following an overall straight trajectory. We chose to focus the search on circular trajectories, centered on the lateral axis, with a variable radius (Fig. 3A),

and for which the robot's body is pointing towards the tangent of the overall trajectory. Straight, forward (or backward) trajectories are still possible with the particular case of an infinite radius. This kind of trajectory is suitable for motion control as many complex trajectories can be decomposed in a succession of circle portions and lines. An illustration of this principle is pictured on figure 3 (D-E-F).

To encourage the population to follow these trajectories, the quality score is set as the angular difference between the arrival orientation and the tangent of the circular trajectory that corresponds to the endpoint (Fig. 3B):

$$\text{quality}(\mathbf{c}) = -|\theta(\mathbf{c})| = -|\alpha(\mathbf{c}) - \beta(\mathbf{c})| \quad (5)$$

4.1.5 Transferability score

The transferability score of a tested controller \mathbf{c}^* is computed as the distance between the controller's endpoint reached in simulation and the one reached in reality:

$$\text{transferability}(\mathbf{c}^*) = -|\mathcal{E}_{\text{simu}} - \mathcal{E}_{\text{real}}| \quad (6)$$

In order to estimate the transferability score of untested controllers, a regression model is trained with the tested controllers and their recorded transferability score. The regression model used is the ν -Support Vector Regression algorithm with linear kernels implemented in the library *libsvm* (Chang and Lin, 2011) (learning parameters are set to default values), which maps a behavioral descriptor ($\text{des}(\mathbf{c})$) with an estimated transferability score ($\mathcal{T}(\text{des}(\mathbf{c}))$). Each controller is described with a vector of Boolean values that describe, for each time-step and each leg, whether the leg is in contact with the ground (the descriptor is therefore a vector of size $N \times 6$, where N is the number of time-steps). This kind of vector is a classic way to describe gaits in legged animals and robots (Raibert, 1986). During the evolutionary process, the algorithm performs 1 transfer every 50 iterations.

4.2 Experiments on the Virtual Robot

This first experiment involves a virtual robot that learns a behavioral repertoire to reach every point in its vicinity. The transferability objective is disabled because the goal of this experiment is to show the benefits of learning simultaneously all the behaviors of the repertoire instead of learning them separately. Using only the simulation allows us to perform more replications and to implement a higher number of control experiments. This experiment also shows how the robot is able to *autonomously*:

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

The TBR-Evolution experiment and the control experiments (described in the next section) are replicated 40 times to gather statistics.

4.2.1 Control Experiments

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

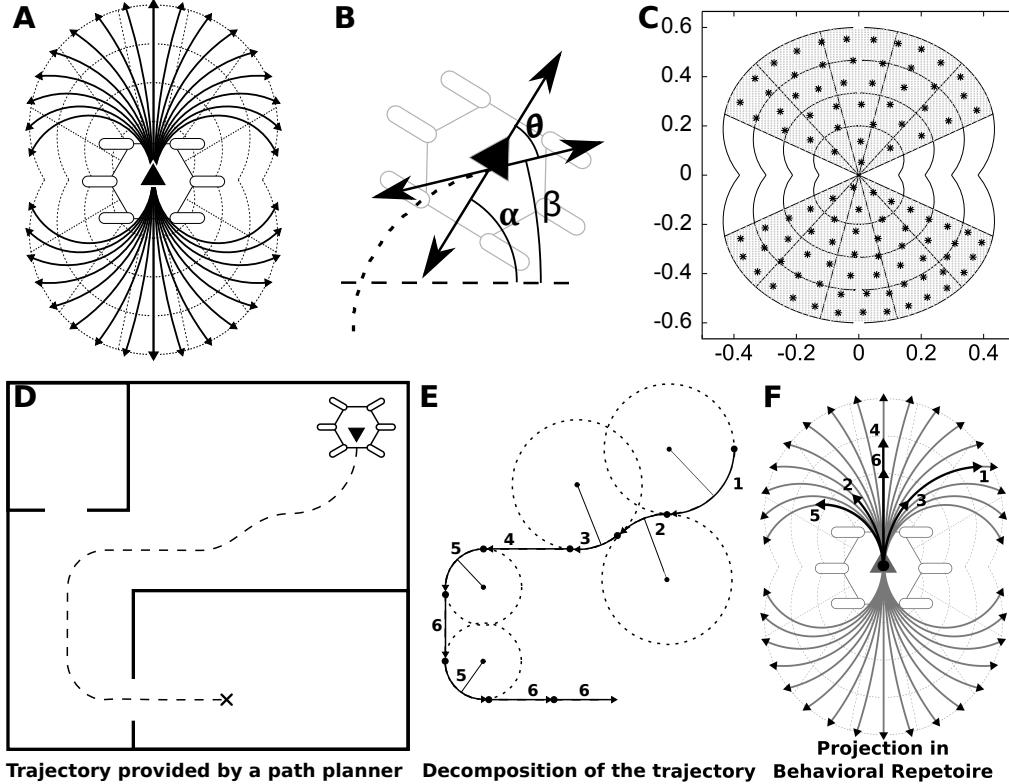


Figure 3: (A) Examples of trajectories following a circle centered on the lateral axis with several radii. (B) Definition of the desired orientation. θ represents the orientation error between α , the final orientation of the robot, and β , the tangent of the desired trajectory. These angles are defined 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). The boundaries of the region are defined by two lines at 60 degrees on each side of the robot. The curved frontier is made of all the reachable points with a curvi-linear abscissa lower than 0.6 meters (these values were set thanks to experimental observations of commonly reachable points). Dots correspond to targets selected for the control experiments. (D-E-F) Illustration of how a behavioral repertoire can be used with a hexapod robot. First, a path planning algorithm computes a trajectory made of lines and portions of circles (LaValle, 2006; Siciliano and Khatib, 2008). Second, to follow this trajectory, the robot sequentially executes the most appropriate behavior in the repertoire (here numbered on E and F). For closed-loop control, the trajectory can be re-computed at each time-step using the actual position of the robot.

We define 100 target points, spread thanks to a K-means algorithm Seber (1984) over the defined region of interest (ROI) of the reachable area (see Fig. 3C). We then execute several multi-objective evolutionary algorithms (NSGA-II Deb et al. (2002)), 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 (quality score described previously) within a radius of 10 cm around the target point and we call this variant “orientation”. The objectives used for the optimization are:

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

We also investigate how the archive management added in TBR-Evolution improves the quality of produced behavioral repertoires. To highlight these improvements, we compared our resulting archives with archives issued from the Novelty Search algorithm (Lehman and Stanley, 2011a) and from the Novelty Search with Local Competition algorithm (Lehman and Stanley, 2011b), as the main difference between these algorithms is archive management procedure. We apply these algorithms on the same task and with the same parameters as in the experiment with our method. We call these experiments “Novelty Search”(NS) and “NS with Local Competition”. For both of these experiments we will analyze both the produced archives and the resulting populations.

For all the experiments we will study the sparseness and the orientation error of the behavioral repertoires generated by each approach. All these measures are done within the region of interest previously defined. The sparseness of the archive is computed by discretizing the ROI with a one centimeter grid (\mathcal{G}), and for each point p of that grid the distance from the nearest individual of the archive (\mathcal{A}) is recorded. The sparseness of the archive is the average of all the recorded distances:

$$\text{sparseness}(\mathcal{A}) = \frac{\sum_{p \in \mathcal{G}} \min_{i \in \mathcal{A}} (\text{distance}(i, p))}{\text{card}(\mathcal{G})} \quad (7)$$

where $\text{card}(\mathcal{G})$ denotes the number of elements in \mathcal{G} .

The quality of the archive is defined as the average orientation error for all the individuals inside the ROI:

$$\text{Orientation Error}(\mathcal{A}) = \frac{\sum_{i \in \mathcal{A} \in \text{ROI}} \theta(i)}{\text{card}(\mathcal{A} \in \text{ROI})} \quad (8)$$

4.2.2 Results

Resulting behavioral repertoires from a typical run of TBR-Evolution and the control experiments are pictured on figures 4, 5 and 9. The endpoints achieved with each controller of the repertoire are spread over the reachable space in a specific manner: they cover both the front and the back of the robot, but less the lateral sides. These limits are not explicitly defined, but they are autonomously discovered by the algorithm.

For the same number of evaluations, the area is less covered with the control experiments (nearest and orientation) than with TBR-Evolution (Fig. 4). With only 100 000 evaluations, this area is about twice larger with TBR-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 Fig. 3C), this is not the case for the “orientation” variant.

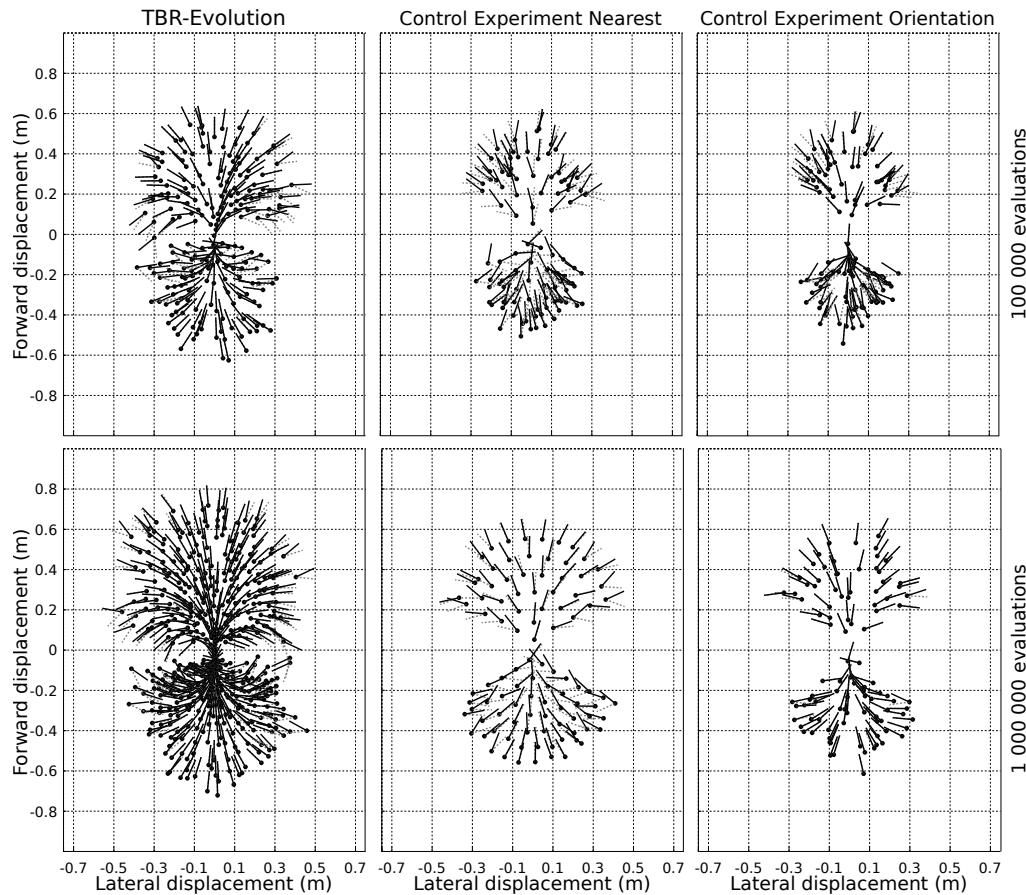


Figure 4: Comparison between the typical results of the TBR-Evolution algorithm, the “nearest”, and the “orientation”. The archives are displayed after 100 000 evaluations (top) and after 1 000 000 evaluations (bottom). Each dot corresponds to the endpoint of a controller. The solid lines represent the final orientation of the robot for each controller, while the gray dashed lines represent the desired orientation. The orientation error is the angle between solid and dashed lines.

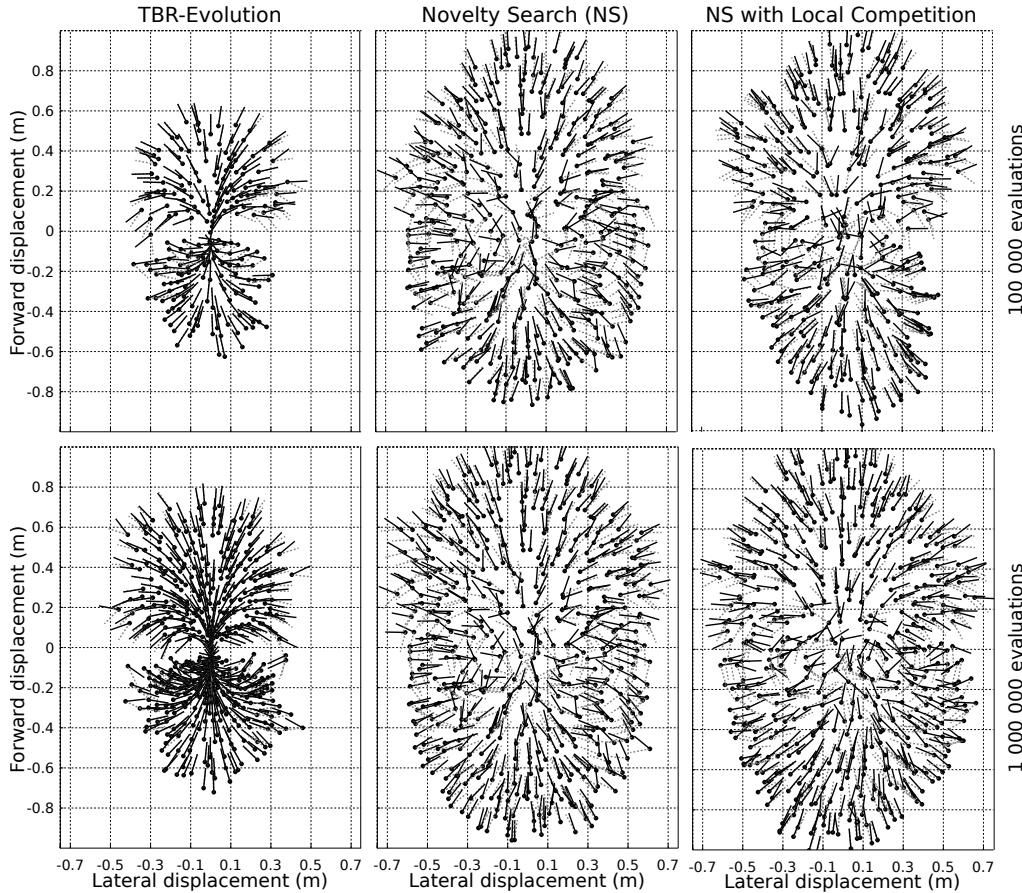


Figure 5: Comparison between the typical results of the TBR-Evolution algorithm, the Novelty Search (NS), and the NS with Local Competition. The archives are displayed after 100 000 evaluations and after 1 000 000 evaluations. Each dot corresponds to the end-point of a controller. The solid lines represent the final orientation of the robot for each controller, while the gray dashed lines represent the desired orientation. The orientation error is the angle between solid and dashed lines.

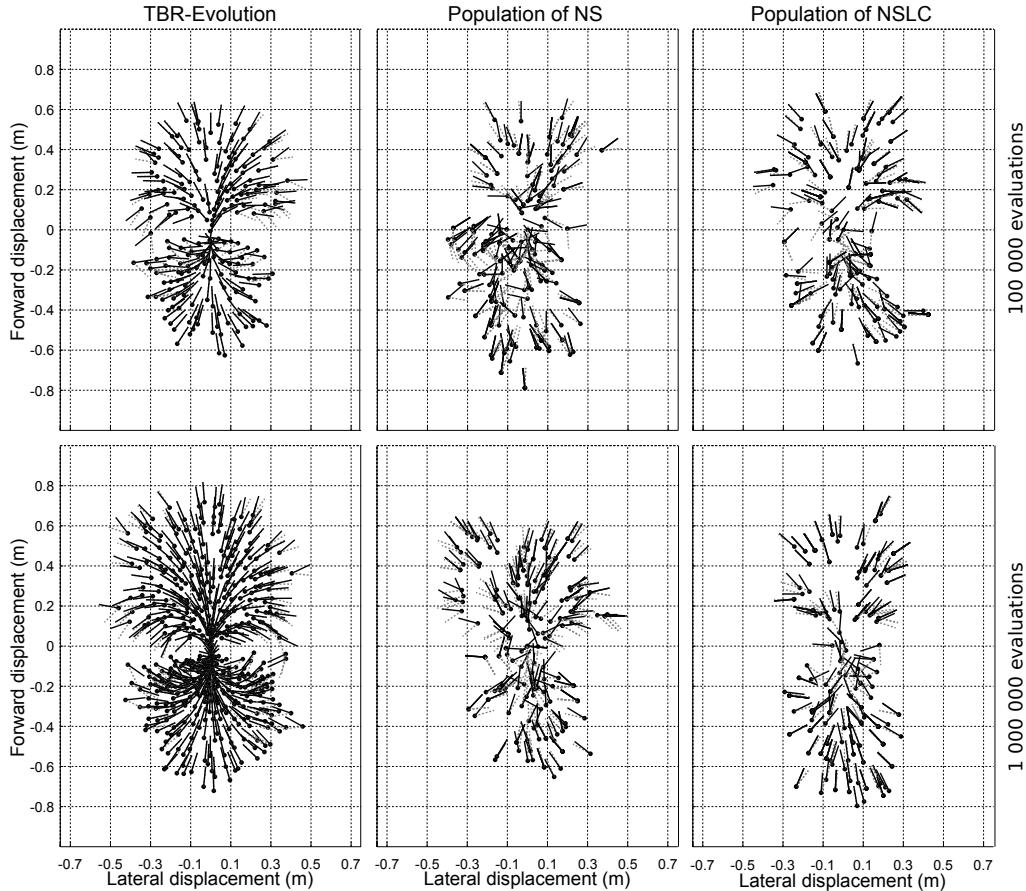


Figure 6: Comparison between typical results of the TBR-Evolution algorithm, the population of Novelty Search, and the population of NS with Local Competition. The archives/populations are displayed after 100 000 evaluations and after 1 000 000 evaluations. Each dot corresponds to the endpoint of a controller. The solid lines represent the final orientation of the robot for each controller, while the gray dashed lines represent the desired orientation. The orientation error is the angle between solid and dashed lines.

The archives produced by Novelty Search and NS with Local Competition both cover a larger space than the TBR-Evolution algorithm (Fig. 5). These results are surprising because all these experiments are based on novelty search and differ only in the way the archive is managed. These results show that TBR-Evolution tends to slightly reduce the exploration abilities of NS and focuses more on the quality of the solutions.

We can formulate two hypotheses to explain this difference in exploration. First, the “local competition” objective may have an higher influence in the TBR-Evolution algorithm than in the NS with Local Competition: in NS with local competition, the individuals from the population are competing against those of the archive; since this archive is not updated if an individual with a similar behavior but a higher performance is encountered, the individuals from the population are likely to always compete against low-performing individuals, and therefore always get a similar local competition score; as a result, the local competition objective is likely to not be very distinctive and most of the selective pressure can be expected to come from the novelty objective. This different selective pressure can explain why NS with local competition explores more than BR-Evolution, and it echoes the observation that the archive obtained with NS and NS with local competition are visually similar (Fig. 5). The second hypothesis is that the procedure used to update the archive may erode the borderline of the archive: if a new individual is located close to the archive’s borderline, and if this individual has a better performance than its nearest neighbor in the archive, then the archive management procedure of TBR-Evolution will replace the individual from the archive with the new and higher-performing one; as a consequence, an individual from the border can be removed in favor of a higher-performing but less innovative individual. This process is likely to repeatedly “erode” the border of the archive and hence discourage exploration. These two hypotheses will be investigated in future work.

The primary purpose of the Novelty Search with Local Competition is to maintain a diverse variety of well adapted solutions in its population, and not in its archive. For this reason, we also plotted the distribution of the population’s individuals for both the Novelty Search and the NS with Local Competition (Fig. 6). After 100,000 evaluations, and at the end of the evolution, the population covers less of the robot’s surrounding than TBR-Evolution. The density of the individuals is not homogeneous and they are not arranged in a particular shape, contrary to the results of TBR-Evolution. In particular, the borderline of the population seems to be almost random.

The density of the archive is also different between the algorithms. The density of the archives produced by TBR-Evolution is higher than the other approaches, while the threshold of novelty (ρ) required to add individuals in the archive is the same. This shows that the archive management of the TBR-Evolution algorithm increases the density of the regions where solutions with a good quality are easier to find. This characteristic allows a better resolution of the archive in specific regions.

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 at the beginning of the “orientation” variant too, but, at the end, the error is negligible for the majority of controllers. The Novelty Search, NS with local competition and the population of the Novelty Search have a larger orientation error, the figures 5 and 6 show that the orientation of the controllers seems almost random. With such repertoire, chaining behaviors on the robot is more complicated than with the TBR-Evolution’s archives, where a vector field is visible. Only the population of the NS with Local Competition seems to show lower orientation error. This illustrates the benefits of the local competition objective on the population’s behaviors.

The TBR-Evolution algorithm consistently leads to very small orientation errors (Fig. 4 and Fig. 9); only few points have a significant error. We find these points in two regions, far from the starting point and directly on its sides. 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 given the controller and the possible parameters (Appendix 4.1). For example the close lateral regions require executing a trajectory with a very high curvature, which cannot be executed with the range of parameters of the controller. Moreover, the behaviors obtained in these regions are most of the time degenerated: they take advantages of inaccuracies in the simulator to realize movement that would not be possible in reality. 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 (Fig. 9): 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 40 runs, of the sparseness and the quality of controllers inside a region of interest (ROI) (Fig. 7, Top). The TBR-Evolution algorithm achieved a low sparseness value with few evaluations. After 100 000 evaluations, it was 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 the “nearest” and the “orientation” experiments, the variation is slower and reaches a significantly higher level of sparseness (p -values = 1.4×10^{-14} with Wilcoxon rank-sum tests). The “orientation” variant of the control experiment exhibits the worst sparseness value ($> 4\text{cm}$). 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). The Novelty Search and the NS with Local Competition experiments follow the same progression (the two lines are indistinguishable) and reach their final value faster than the TBR-Evolution algorithm. As our algorithm can increase the density of controller in particular regions, at the end of the evolution, the final value of sparseness of TBR-Evolution is better than all the control experiments. The sparseness of the populations of Novelty Search and NS with Local Competition are indistinguishable too, but also constant over all the evolution and larger than all the tested algorithms, mainly because of the uneven distribution of their individuals (fig. 6)

From the orientation point of view (Fig. 7, bottom), our approach needs few evaluations to reach a low error value (less than 5 degrees after 100 000 evaluations and less than 1.7 degrees at the end of the evolutionary process). The variation of the “orientation” control experiment is slower and needs 750 000 evaluations to cross the curve of TBR-Evolution. At the end of the experiment this variant reaches a significantly lower error level (p -values = 3.0×10^{-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 (greater than 15 degrees, p -values = 1.4×10^{-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. The Novelty Search and the NS with Local Competition experiments lead to orientation errors that are very high and almost constant over all

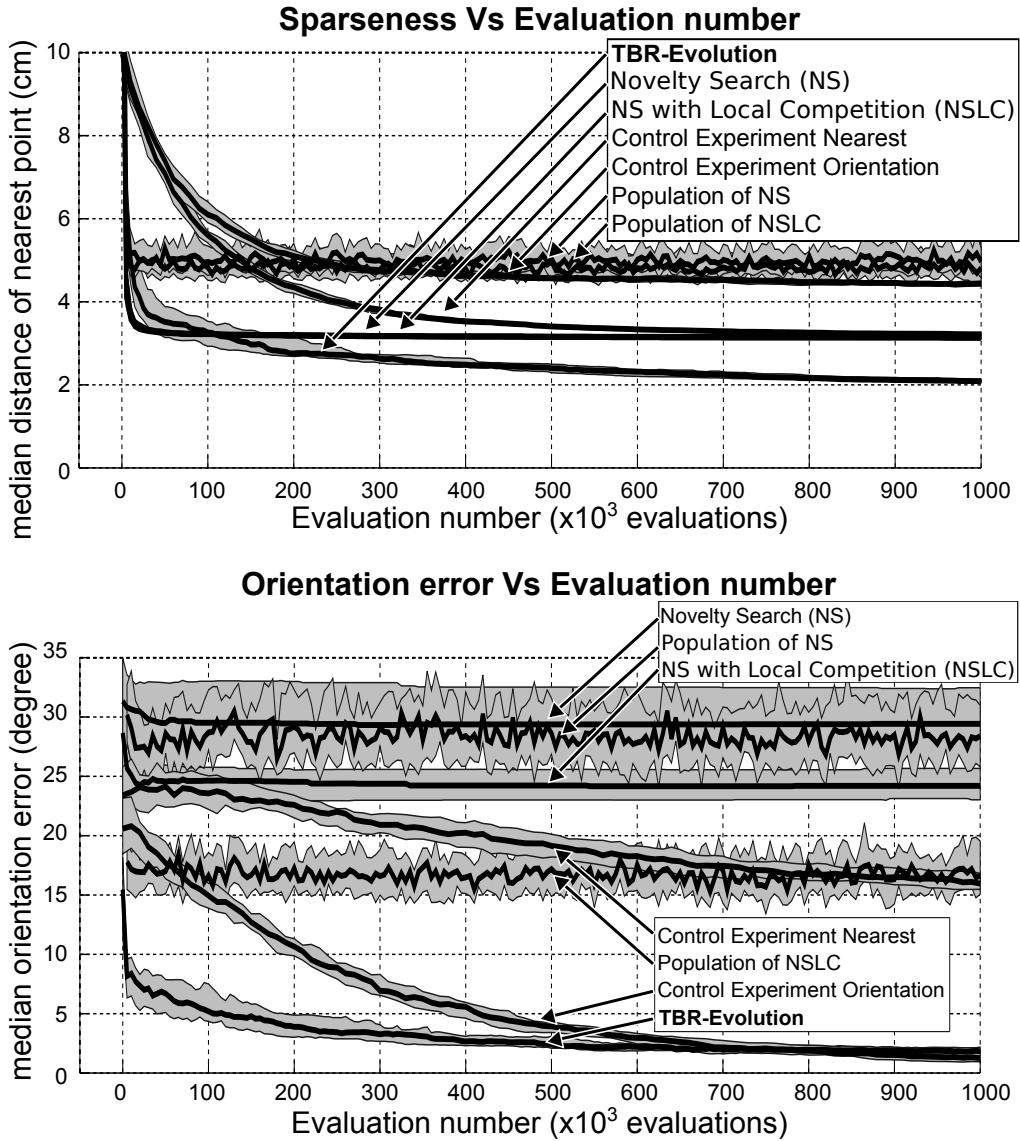


Figure 7: (Top) Variation of the sparseness of the controller repertoire. For each point of a one centimeter grid inside the ROI (Fig. 3), the distance from the nearest controller is computed. The sparseness value is the average of these distances. This graph plots the first three quartiles of the sparseness computed with 40 runs for each algorithm. (Bottom) Variation of the median of the orientation error over all the controllers inside the region of interest. This graph also plots the three first quartiles (25%, 50%, 75%) computed with 40 runs for each algorithm.

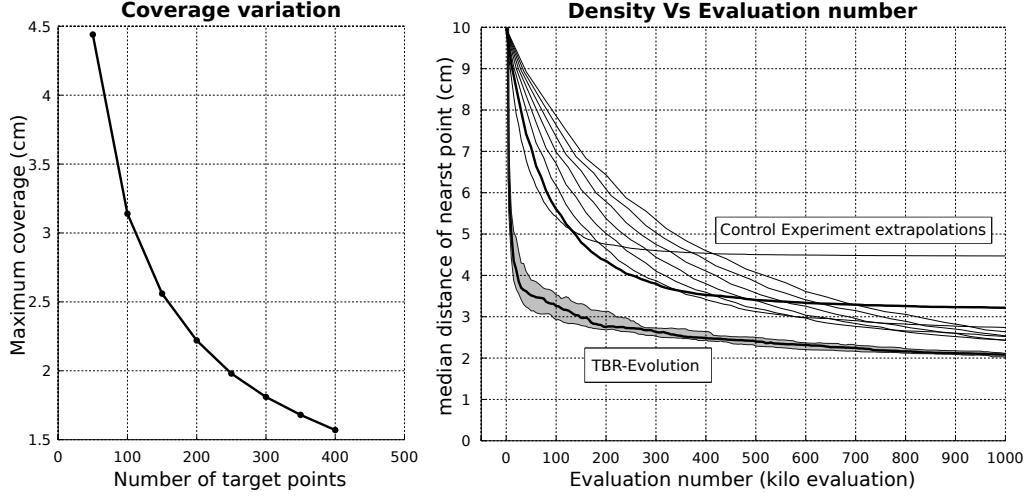


Figure 8: (Left) Theoretical sparseness of the control experiments according to the number of points. With more points, the sparseness value will be better (lower). (Right) Extrapolations of the variation of the sparseness for the “nearest” variant of the control experiment according to different number of targets. Each line is an extrapolation of the variation of the sparseness of the “nearest variant”, which are based on a number of points starting from 50 to 400, with a 50 points step. The variation of TBR-Evolution is also plotted for comparison.

the evolution. These results come from the archive management of these algorithms which do not substitute individuals when a better one is found. The archive of these algorithms only gathers the first encountered behavior of each reached point. The orientation error of the NS with Local Competition is lower than the Novelty Search because the local competition promotes behavior with a good orientation error (compared to their local niche) in the population, which has an indirect impact on the quality of the archive but not enough to reach a low error level. The same conclusion can be drawn with the population of these two algorithms: while the populations of the Novelty Search have a similar orientation error than its archives, the populations of the NS with Local Competition have a lower orientation error than its archives.

With the sets of reference points, we can compute the theoretical minimal sparseness value of the control experiments (Fig. 8, 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 number of evaluations. Thanks to these values we can extrapolate the variation of the sparseness according to the number of points. For example, with 200 targets, we can predict that the final value of the sparseness will be 2.22 and thus we can scale the graph of our control experiment to fit this prediction. Increasing the number of targets will necessarily increase the number of evaluations, for example using 200 targets will double the number of evaluations. Following this constraint, we can also scale the temporal axis of our control experiment. We can thus extrapolate the sparseness of the archive with regard to the number of target, and compare it to the sparseness of the archive generated with TBR-Evolution.

The extrapolations (Fig. 8, right) show higher sparseness values compared to TBR-

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 7 and 8 demonstrate how TBR-Evolution is better both in the sparseness and in the orientation aspects compared than the 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)

4.3 Experiments on the Physical Robot

In this second set of experiments, we apply the TBR-Evolution algorithm on a physical hexapod robot (see Fig. 1 left). The transferability component of the algorithm allows it to evolve the behavioral repertoire with a minimum of evaluation on the physical robot. For this experiment, 3000 generations are performed and we execute a transfer (evaluation of one controller on the physical robot) every 50 generations, leading to a total of 60 transfers. The TBR-Evolution experiments and the reference experiments are replicated 5 times to gather statistics².

4.3.1 Reference Experiment

In order to compare the learning speed of the TBR-Evolution algorithm, we use a reference experiment where only one controller is learned. For this experiment, we use the NSGA-II algorithm with the transferability approach to learn a controller that reaches a predefined target. The target is situated 0.4m in front and 0.3m to the right: a point not as easy to be accessed as going only straight forward, and not as hard as executing a U-turn. It represents a good difficulty trade-off and thus allows us to extrapolate the performances of this method to more points.

The main objective is the distance ($\text{Distance}(\mathbf{c})$) between the endpoint of the considered controller and the target. The algorithm also optimizes the estimated transferability value ($\hat{\mathcal{T}}(\text{des}(\mathbf{c}))$) and the orientation error ($\text{perf}(\mathbf{c})$) with the same definition as in the TBR-Evolution algorithm:

$$\text{minimize} \begin{cases} \text{Distance}(\mathbf{c}) \\ \hat{\mathcal{T}}(\text{des}(\mathbf{c})) \\ \text{perf}(\mathbf{c}) \end{cases}$$

To update the transferability function, we use the same transfer frequency as in TBR-Evolution experiments (every 50 generations). Among the resulting trade-offs, we select as final controller the one that arrives closest to the target among those with an estimated transferability $\hat{\mathcal{T}}(\text{des}(\mathbf{c}))$ less than 0.10m. This represents a distance between the endpoint reached in simulation and the one reached in reality lower than 10 cm.

4.3.2 Results

After 3000 iterations and 60 transfers, TBR-Evolution generates a repertoire with a median number of 375 controllers (min = 352, max = 394). This is achieved in approximately 2.5 hours. One of these repertoires is pictured in figure 10, left. The distribution of the controllers' endpoints follows the same pattern as in the virtual experiments:

²Performing statistical analysis with only 5 runs is difficult but it still allows us to understand the main tendencies. The current set of experiments (5 runs of TBR-Evolution and the control experiment) requires more than 30 hours with the robot and it is materially challenging to use more replications.

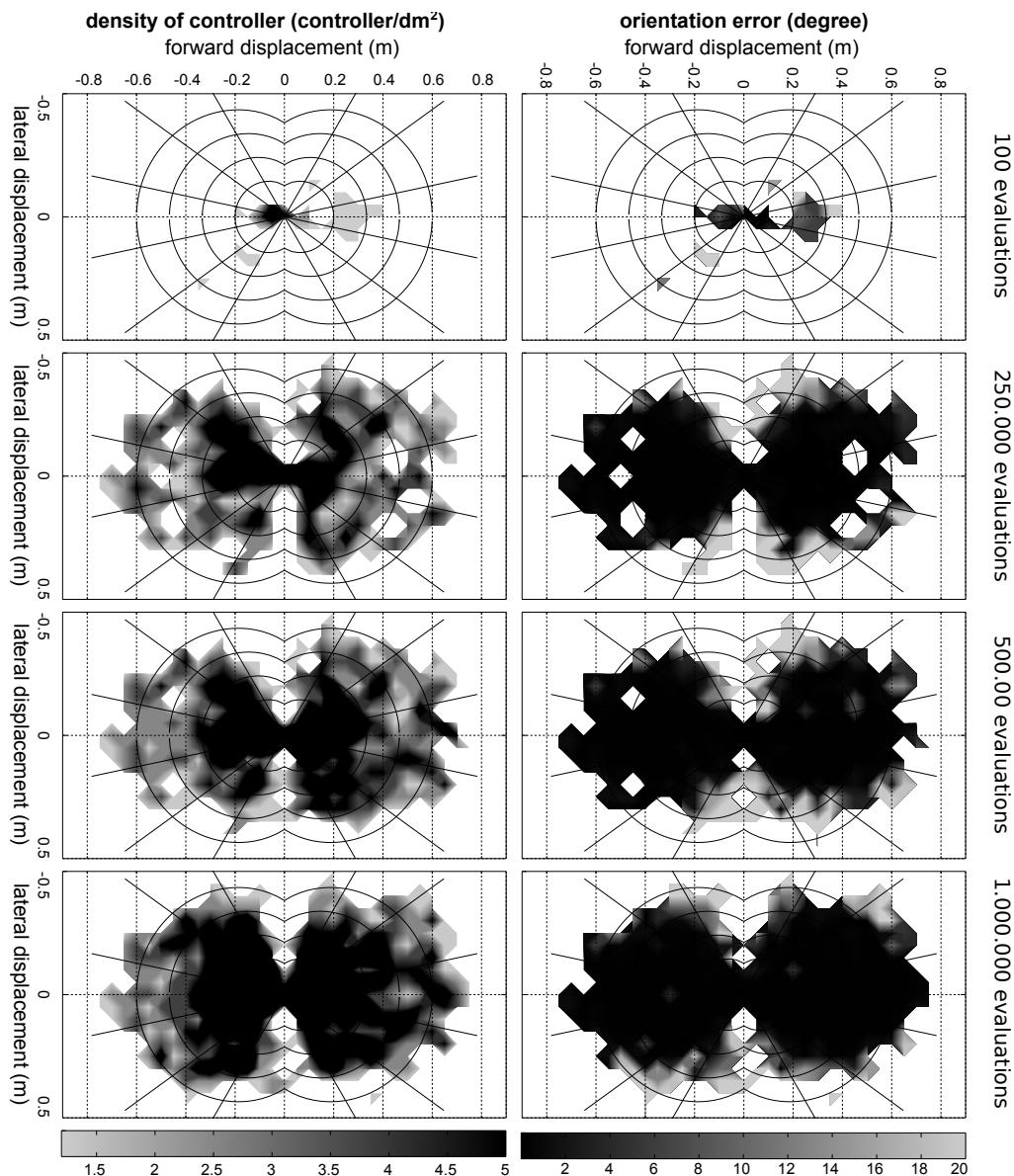


Figure 9: (Left) Variation of density of controller (number of controllers per dm^2). (Right) Variation of the orientation error (given by the nearest controller) along a typical run in simulation.

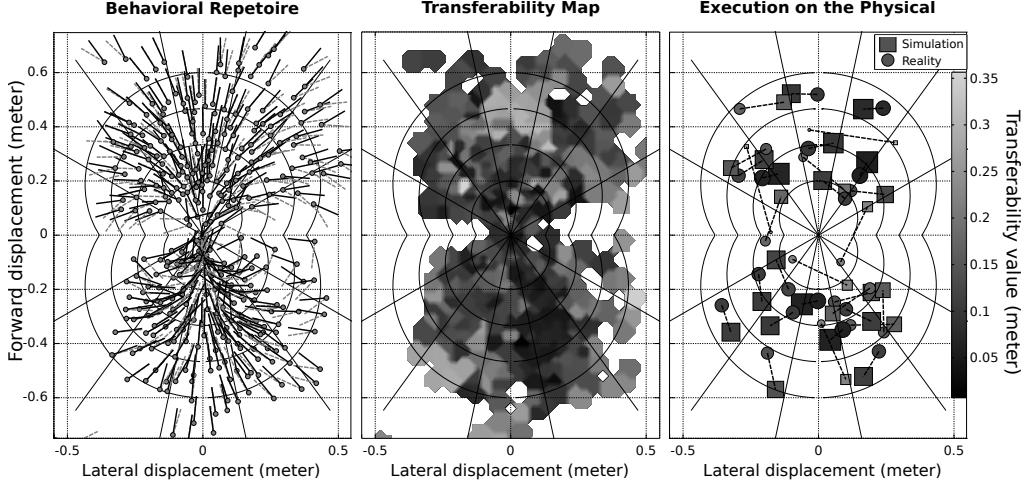


Figure 10: Typical repertoire of controllers obtained with the TBR-Evolution algorithm. (Left) The dots represent the endpoints of each controller. The solid lines are the final orientations of the robot while the dashed ones are the desired orientations. The angle between these two lines is the orientation error. (Center) Transferability map. For each point of the reachable space, the estimated transferability of the nearest controller, within a radius of 5cm, is pictured. (Right) Execution on the physical robot. The 30 selected controllers are pictured with square and their actual endpoint with circles. The size and the color of the markers are proportional to their accuracy. To select the tested controllers, the reachable space is split into 30 regions. Their boundaries are defined by two lines at 60 degrees on each side of the robot and by two curved frontiers that regroup all reachable points with a curvi-linear abscissa between 0.2 and 0.6 m. These regions are then segmented into 15 parts for both the front and the rear of the robot. All of these values are set from experimental observations of commonly reachable points.

they cover both the front and the rear of the robot, but not the lateral sides. Here again, these limits are not explicitly defined, they are autonomously discovered by the algorithm.

Similarly to the experiments on the virtual robot, the majority of the controllers have a good final orientation and only the peripheries of the repertoire have a distinguishable orientation error. TBR-Evolution successfully pushes the repertoire of controllers towards controllers with a good quality score and thus following the desired trajectories.

From these results we can draw the same conclusion as with the previous experiment: the difficulty of accessing peripheral regions explains the comparatively poor performances of controllers from these parts of archive. The large distance to the starting point or the complexity of the required trajectory meets the limits of the employed controllers.

The transferability map (Fig. 10, center) shows that the majority of the controllers have an estimated value lower than 15cm (dark regions). Nevertheless, some regions are deemed non-transferable (light regions). These regions are situated in the peripheries too, but are also in circumscribed areas inside of the reachable space. Their occurrence in the peripheries have the same reasons as for the orientation (section 4.2),

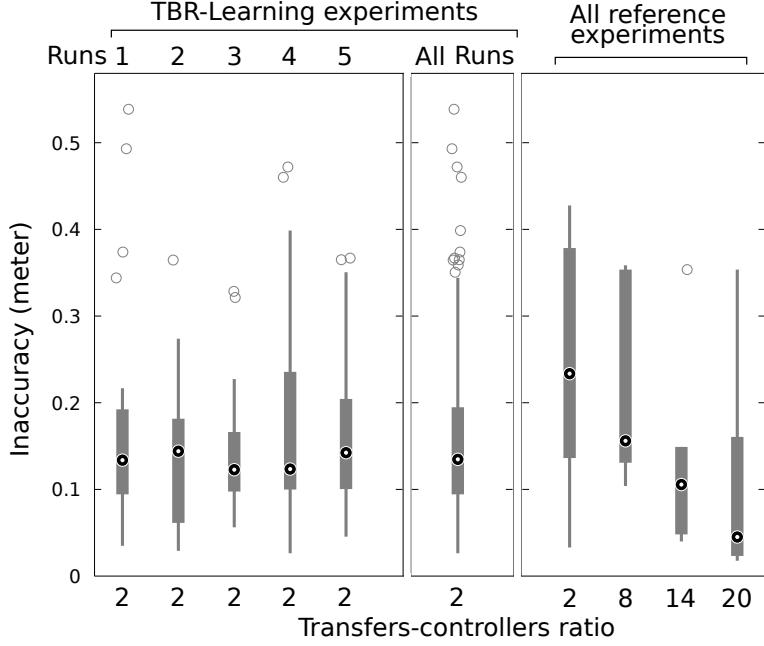


Figure 11: Accuracy of the controllers. The accuracy is measured as the distance between the endpoint reached by the *physical robot* and the one reached in simulation (30 points for each run, see text). The results of the TBR-Evolution experiments are, for each run, separately pictured (Left) and also combined for an overall point of view (Center). The performances of the reference experiments are plotted according to the number of transfers performed (Right). In both cases, one transfer is performed every 50 iterations of the algorithm.

but those inside the reachable space show that the algorithm failed to find transferable controllers in few specific regions. This happens when the performed transfers do not allow the algorithm to infer transferable controllers. To overcome this issue, different selection heuristics and transfer frequencies will be considered in future work.

In order to evaluate the hundreds of behaviors contained in the repertoires on the physical robot, we select 30 controllers in each repertoire of the 5 runs. The selection is made by splitting the space into 30 areas (Fig. 10) and selecting the controllers with the best estimated transferability in each area.

Most of these controllers have an actual transferability value lower than 15 cm (Fig. 11, left), which is consistent with the observations of the transferability map (Fig. 10, center) and not very large once taken into consideration the SLAM precision, the size of the robot and the looseness in the joints. Over all the runs, the median accuracy of the controllers is 13.5 cm (Fig. 11, center). Nevertheless, every run presents outliers, i.e. controllers with a very bad actual transferability value, which originate from regions that the transferability function does not correctly approximate.

In order to compare the efficiency of our approach to the reference experiment, we use the “transfers-controllers ratio”, that is the number of performed transfers divided by the number of produced controllers at the end of the evolutionary process. For instance, if we reduce the produced behavioral repertoires to the 30 tested controllers,

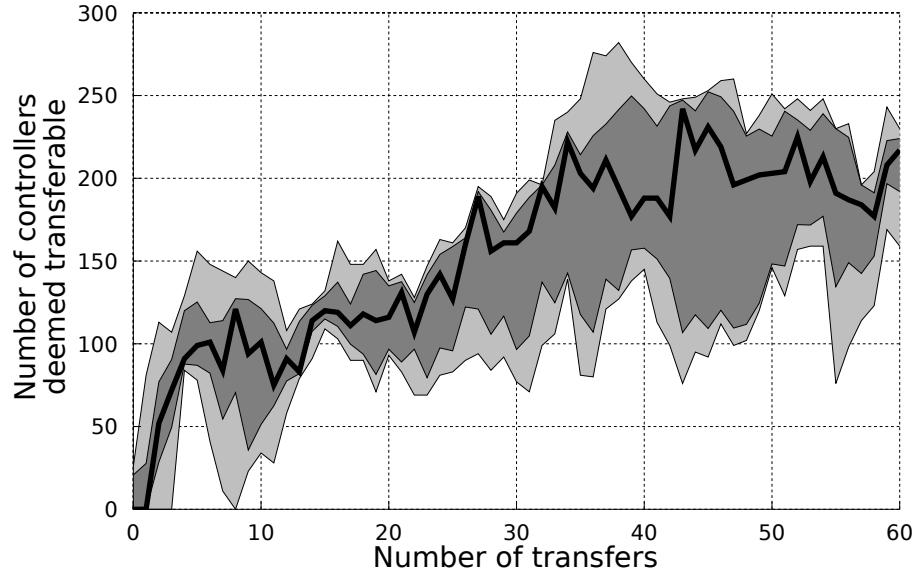


Figure 12: Evolution of the number of controllers deemed transferable. At each transfer (i.e. every 50 iterations), the number of controllers with an estimated transferability lower than 15cm is pictured for all the 5 runs. The bold black line represents the median value, the dark region the first and third quartile and the light one the lower and upper bounds. The variability of the curve is due to the periodic transfers, which update the transferability function and thus the estimated transferability values.

this ratio is equal to $60/30 = 2$ for the TBR-Evolution experiments, since we performed 60 transfers.

The performances of the control experiments depend on the number of performed transfers (Fig. 11, right) and thus on this ratio. For an equal ratio, the reference experiments are 74% less accurate than TBR-Evolution (13.5 cm vs. 23.4 cm, p-value= 0.12 with the Wilcoxon ranksum test), while the accuracies of both experiments are not statistically different (13.5 cm vs. 15.6 cm and 10.6 cm, p-value= 0.23 and respectively 0.35) if the reference algorithm uses from 8 to 14 transfers to learn one controller (i.e. a process 4 to 7 times longer). The reference experiment only takes advantage of its target specialisation when 20 transfers are performed. With a transfers-controllers ratio equals to 20, the accuracy of the reference controllers outperforms the controllers generated with the TBR-Evolution algorithm (13.5 cm vs. 4.5 cm, p-value= 0.06). Nevertheless, with such a high ratio, the reference experiment only generates 3 controllers, while our approach generates 30 of them with the same running time (60 transfers and 3000 generations).

We previously only considered the 30 post evaluated controllers, whereas TBR-Evolution generates several hundreds of them. After 60 transfers, the repertoires contain a median number of 217 controllers that have an estimated transferability lower than 0.15 m (Fig. 12). The previous results show that more than 50% of the tested controllers have an actual transferability value lower than 0.15 m and 75% lower than 0.20 m. We can consequently extrapolate that between 100 and 150 controllers are exploitable in a typical behavioral repertoire. Once taking into consideration all these controllers, the transfers-controllers ratio of the TBR-Evolution experiments falls be-

tween 0.4 and 0.6 and thus our approach is about 25 times faster than the reference experiment, for a similar accuracy.

5 Conclusion and Discussion

To our knowledge, TBR-Evolution is the first algorithm designed to generate a large number of efficient gaits without requiring to learn each of them separately, or to test complex controllers for each direction. In addition, our experiments only rely on internal, embedded measurements, which is critical for autonomy, but not considered in most previous studies (e.g., Kohl and Stone (2004); Zykov et al. (2004); Chernova and Veloso (2004); Yosinski et al. (2011); Mahdavi and Bentley (2006)).

We evaluated our method on two experiments, one in simulation and one with a physical hexapod robot. With these experiments, we showed that, thanks to its ability to recycle solutions usually wasted by classic evolutionary algorithm, TBR-Evolution generate behavioral repertoires faster than by evolving each solution separately. We also showed that the archive management allows it to generate behavioral repertoire with a significantly higher quality than the Novelty Search algorithm (Lehman and Stanley, 2011a).

With the TBR-Evolution algorithm, our physical hexapod robot was able to learn several hundreds of controllers with only 60 transfers of 3 seconds on the robot, which was achieved in 2.5 hours (including computation time for evolution and the SLAM algorithm). The repartition of these controllers over all the reachable space has been autonomously inferred by the algorithm according to the abilities of the robot. Our experiments also showed that our method is about 25 times faster than learning each controller separately.

Overall, these experiments show that *the TBR-Evolution algorithm is a powerful method for learning multiple tasks with only several dozens of tests on the physical robot*. Figure 13 and the supplementary video illustrate the resulting ability of the robot to walk in every direction. In the footsteps of Novelty Search, this new algorithm thus highlights that evolutionary robotics can be more than black-box optimization (Doncieux and Mouret, 2014): evolution can simultaneously optimize in many niches, each of them corresponding to a different, but high-performing, behavior.

In future work, we plan to investigate the generation of behavioral repertoires in an environment with obstacles. Selecting the transfers according to the presence of obstacles might enable the robot to avoid them during the learning process. This ability is rarely considered in this kind of learning problem: most of the time, the robot is placed in an empty space or manually replaced in a starting point (for example Kohl and Stone (2004); Zykov et al. (2004); Berenson et al. (2005), and the present work). This would be a step forward in autonomous learning in robotics.

Both the Transferability approach and the Novelty Search with Local Competition do not put any assumption on the type of controller or genotype employed. For example, the Transferability approach has been used to evolve the parameters of Central Pattern Generators (Oliveira et al., 2013) or those of an Artificial Neural Networks (Koos et al., 2013b), and the Novelty Search algorithm has been employed on plastic artificial neural encoded with NEAT (Risi et al., 2010; Lehman and Stanley, 2011a) and on graph-based virtual creatures (Lehman and Stanley, 2011b). Similarly, since the TBR-Evolution is the combination of these two algorithms, it can also be used with any type of genotype or controller. In future work, we will therefore investigate more sophisticated genotypes and phenotypes like, for instance, neural networks encoded with HyperNEAT (Stanley et al., 2009; Clune et al., 2011; Tarapore and Mouret,

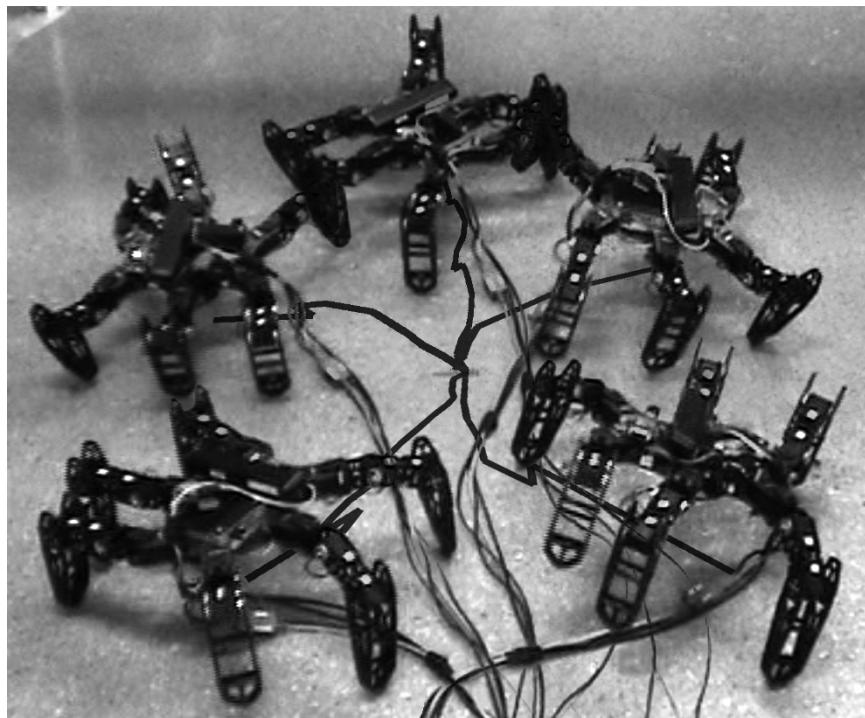


Figure 13: Illustration of the results. These 5 typical trajectories correspond to controllers obtained with TBR-Evolution as recorded by the SLAM algorithm. The supplementary video shows a few other examples of controllers.

2014a,b), or oscillators encoded by compositional pattern-producing networks (CPPN), like SUPGs (Morse et al., 2013; Tarapore and Mouret, 2014a,b). Nevertheless, such advanced controllers can use feedback to change their behavior according to their sensors, and understanding how feedback-driven controllers and a repertoire-based approach can be elegantly combined is an open question.

The TBR-Evolution puts also no assumption on the type of robot and it would be interesting to see the abilities of the algorithm on more challenging robots like quadrupedal or bipedal robots, where the stability is more critical than with the hexapod robot.

The ability of TBR-Evolution to autonomously infer the possible actions of the robot makes this algorithm a relevant tool for developmental robotics (Lungarella et al., 2003). With our methods, the robot progressively discovers its abilities and then perfects them. This process is similar to the “artificial curiosity” algorithms in developmental robotics (Barto et al., 2004; Oudeyer, 2004), which make robots autonomously discover their abilities by exploring their behavioral space. It will be relevant to study the links between these approaches and our algorithm, which come from different branches of artificial intelligence. For example, can we build a behavioral repertoire thanks to the artificial curiosity? Or, can we see the novelty search aspects of TBR-Evolution like a curiosity process? Which of these two approaches is less affected by the curse of dimensionality?

6 Acknowledgments

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

APPENDIX

The source-code of our experiments and a supplementary video can be downloaded from: http://pages.isir.upmc.fr/evorob_db

- Parameters used for the experiments on the virtual robot:
 - TBR-Evolution, Novelty Search and NS with Local Competition experiments:
 - * Population size: 100 individuals
 - * Number of generations: 10 000
 - * Mutation rate: 10% on each parameters
 - * Crossover: disabled
 - * ρ : 0.10 m
 - * ρ variation: none
 - * k : 15
 - “Nearest” and “Orientation” control experiments:
 - * Population size : 100 individuals
 - * Number of generations : 50 0000 (100 * 500)
 - * Mutation rate : 10% on each parameters
 - * Crossover : disabled
- Parameters used for the experiments on the physical robot:
 - TBR-Evolution and the control experiment:

- * Population size: 100 individuals
- * Number of generations: 3 000 generations
- * Mutation: 10% on each parameters
- * Crossover: disabled
- * ρ : 0.10 m
- * ρ variation: none
- * Transfer period: 50 iterations
- * τ : -0.05 m

References

- Barto, A., Singh, S., and Chentanez, N. (2004). Intrinsically motivated learning of hierarchical collections of skills. *Proc. ICDL 2004*.
- Berenson, D., Estevez, N., and Lipson, H. (2005). Hardware evolution of analog circuits for in-situ robotic fault-recovery. In *Proc. of NASA/DoD Conference on Evolvable Hardware*, pages 12–19.
- Bongard, J., Zykov, V., and Lipson, H. (2006). Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121.
- Bongard, J. C. (2013). Evolutionary robotics. *Communications of the ACM*, 56(8):74–83.
- Calandra, R., Seyfarth, A., Peters, J., and Deisenroth, M. P. (2014). An experimental comparison of bayesian optimization for bipedal locomotion. In *Proceedings of 2014 IEEE International Conference on Robotics and Automation (ICRA)*.
- Chang, C. and Lin, C. (2011). Libsvm: a library for support vector machines. *ACM Trans. on Intelligent Systems and Technology*, 2(3):27.
- Chernova, S. and Veloso, M. (2004). An evolutionary approach to gait learning for four-legged robots. In *Proc. of IEEE/RSJ IROS*.
- Clune, J., Stanley, K., Pennock, R., and Ofria, C. (2011). On the performance of indirect encoding across the continuum of regularity. *IEEE Trans. on Evolutionary Computation*, 15(3):346–367.
- Cully, A. and Mouret, J.-B. (2013). Behavioral repertoire learning in robotics. In *Proc of GECCO*, pages 175–182. ACM.
- Currie, K. and Tate, A. (1991). O-plan: the open planning architecture. *Artificial Intelligence*, 52(1):49–86.
- de Garis, H. (1990). Genetic programming: Building nanobrains with genetically programmed neural network modules. In *International Joint Conference on Neural Networks (IJCNN)*, pages 511–516. IEEE.
- Dean, T. and Wellman, M. (1991). *Planning and control*. Morgan Kaufmann Publishers Inc.
- Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. on Evolutionary Computation*, 6(2):182–197.

- Doncieux, S. and Mouret, J.-B. (2014). Beyond black-box optimization: a review of selective pressures for evolutionary robotics. *Evolutionary Intelligence*, pages 1–23.
- Endres, F., Hess, J., Engelhard, N., Sturm, J., Cremers, D., and Burgard, W. (2012). An evaluation of the RGB-D SLAM system. In *Proc. IEEE ICRA*.
- Filliat, D., Kodjabachian, J., and Meyer, J.-A. (1999). Incremental evolution of neural controllers for navigation in a 6-legged robot. In *Proc. of the Fourth International Symposium on Artificial Life and Robots*.
- Gruau, F. (1994). Automatic definition of modular neural networks. *Adaptive behavior*, 3(2):151–183.
- Hornby, G., Takamura, S., Yamamoto, T., and Fujita, M. (2005). Autonomous evolution of dynamic gaits with two quadruped robots. *IEEE Trans. on Robotics*, 21(3):402–410.
- Ijspeert, A. J. (2008). Central pattern generators for locomotion control in animals and robots: a review. *Neural Networks*, 21(4):642–653.
- Ijspeert, A. J., Crespi, A., Ryczko, D., and Cabelguen, J.-M. (2007). From swimming to walking with a salamander robot driven by a spinal cord model. *science*, 315(5817):1416–1420.
- Jacobs, R., Jordan, M., Nowlan, S., and Hinton, G. (1991). Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87.
- Jakobi, N., Husbands, P., and Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. *Proceedings of the European Conference on Artificial Life (ECAL)*, pages 704–720.
- Kimura, H., Yamashita, T., and Kobayashi, S. (2001). Reinforcement learning of walking behavior for a four-legged robot. In *Proc. of Conference on Decision and Control*, volume 1, pages 411–416. IEEE.
- Kodjabachian, J. and Meyer, J.-A. (1998). Evolution and development of neural controllers for locomotion, gradient-following, and obstacle-avoidance in artificial insects. *Neural Networks, IEEE Transactions on*.
- Kohl, N. and Stone, P. (2004). Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proc. of IEEE ICRA*.
- Koos, S., Cully, A., and Mouret, J.-B. (2013a). Fast damage recovery in robotics with the t-resilience algorithm. *The International Journal of Robotics Research*, 32(14):1700–1723.
- Koos, S., Mouret, J.-B., and Doncieux, S. (2013b). The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Trans. on Evolutionary Computation*, pages 122–145.
- Kuffner, J. J. and LaValle, S. M. (2000). Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE.
- LaValle, S. M. (2006). *Planning algorithms*. Cambridge university press.

- Lehman, J. and Stanley, K. (2011a). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2).
- Lehman, J. and Stanley, K. (2011b). Evolving a diversity of virtual creatures through novelty search and local competition. In *Proc. of GECCO*, pages 211–218. ACM.
- Lewis, M. A., Fagg, A. H., and Solidum, A. (1992). Genetic programming approach to the construction of a neural network for control of a walking robot. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pages 2618–2623. IEEE.
- Lizotte, D. J., Wang, T., Bowling, M. H., and Schuurmans, D. (2007). Automatic gait optimization with gaussian process regression. In *Proceedings of the the International Joint Conference on Artificial Intelligence (IJCAI)*, volume 7, pages 944–949.
- Lungarella, M., Metta, G., Pfeifer, R., and Sandini, G. (2003). Developmental robotics: a survey. *Connection Science*, 15(4):151–190.
- Mahdavi, S. and Bentley, P. (2006). Innately adaptive robotics through embodied evolution. *Autonomous Robots*, 20(2):149–163.
- Morse, G., Risi, S., Snyder, C. R., and Stanley, K. O. (2013). Single-unit pattern generators for quadruped locomotion. In *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*, pages 719–726. ACM.
- Mouret, J.-B. and Doncieux, S. (2010). Sferes_{v2}: Evolvin' in the Multi-Core World. In *Proc. of IEEE CEC*, pages 4079–4086.
- Mouret, J.-B., Doncieux, S., and Meyer, J.-A. (2006). Incremental evolution of target-following neuro-controllers for flapping-wing animats. *From Animals to Animats 9*.
- Mouret, J.-B., Koos, S., and Doncieux, S. (2012). Crossing the reality gap: a short introduction to the transferability approach. In *Proceedings of the workshop "Evolution in Physical Systems"*, ALIFE.
- Oliveira, M. A. C., Doncieux, S., Mouret, J.-B., and Santos, C. P. (2013). Optimization of humanoid walking controller: Crossing the reality gap. In *Proceedings of Humanoids*.
- Oudeyer, P.-Y. (2004). Intelligent adaptive curiosity: a source of self-development. *Proc. of the Fourth International Workshop on Epigenetic Robotics*, 117:127–130.
- Pfeifer, R. and Bongard, J. (2007). *How the body shapes the way we think: a new view of intelligence*. MIT press.
- Pfeifer, R., Lungarella, M., and Iida, F. (2007). Self-organization, embodiment, and biologically inspired robotics. *Science*, 318(5853):1088–1093.
- Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). ROS: an open-source robot operating system. In *Proc. of ICRA's workshop on Open Source Software*.
- Raiert, M. H. (1986). Legged robots. *Communications of the ACM*, 29(6):499–514.
- Risi, S., Hughes, C. E., and Stanley, K. O. (2010). Evolving plastic neural networks with novelty search. *Adaptive Behavior*, 18(6):470–491.

- Russell, S., Norvig, P., and Davis, E. (2010). *Artificial intelligence: a modern approach*. Prentice hall Upper Saddle River, NJ.
- Samuelson, E. and Glette, K. (2014). Some distance measures for morphological diversification in generative evolutionary robotics. In *Proceedings of the 16th Annual conference on Genetic and evolutionary computation (GECCO)*. ACM. To appear.
- Schaal, S. (2003). Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *2nd International Symposium on Adaptive Motion of Animals and Machines*.
- Schapire, R. E. (1990). The strength of weak learnability. *Machine learning*, 5(2):197–227.
- Seber, G. (1984). *Multivariate observations*, volume 41. Wiley New York.
- Siciliano, B. and Khatib, O. (2008). *Springer handbook of robotics*. Springer.
- Sproewitz, A., Moeckel, R., Maye, J., and Ijspeert, A. (2008). Learning to move in modular robots using central pattern generators and online optimization. *The International Journal of Robotics Research*, 27(3-4):423–443.
- Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212.
- Tarapore, D. and Mouret, J.-B. (2014a). Comparing the evolvability of generative encoding schemes. In *Proceedings of ALife 14*, pages 55–62. MIT Press.
- Tarapore, D. and Mouret, J.-B. (2014b). Evolvability signatures of generative encodings: beyond standard performance benchmarks. *arXiv preprint arXiv:1410.4985*.
- Tedrake, R., Zhang, T., and Seung, H. (2005). Learning to walk in 20 minutes. In *Proc. of Yale workshop on Adaptive and Learning Systems*.
- Valsalam, V. K. and Miikkulainen, R. (2008). Modular neuroevolution for multilegged locomotion. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 265–272. ACM.
- Wilson, M. (2002). Six views of embodied cognition. *Psychonomic bulletin & review*, 9(4):625–636.
- Yosinski, J., Clune, J., Hidalgo, D., Nguyen, S., Zagal, J., and Lipson, H. (2011). Evolving Robot Gaits in Hardware: the HyperNEAT Generative Encoding Vs. Parameter Optimization. *Proc. of ECAL*.
- Zykov, V., Bongard, J., and Lipson, H. (2004). Evolving dynamic gaits on a physical robot. In *Proc. of GECCO, Late Breaking Paper*.