# Fast Damage Recovery in Robotics with the T-Resilience Algorithm

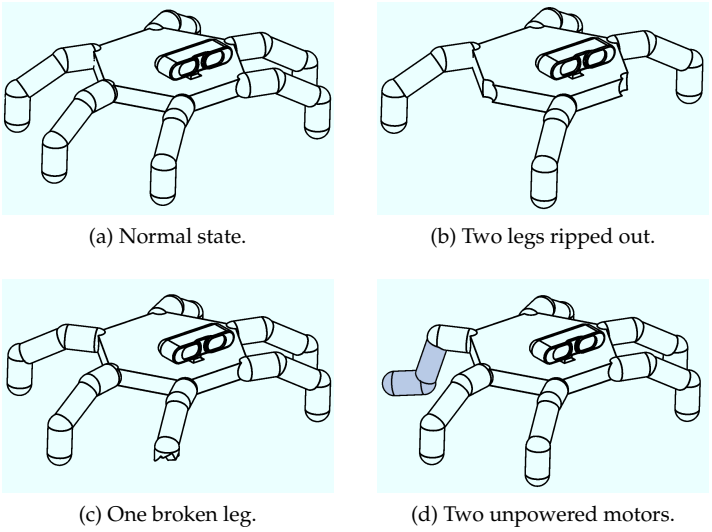
Sylvain Koos, Antoine Cully, Jean-Baptiste Mouret

S. Koos, A. Cully and J.B. Mouret. Fast Damage Recovery in Robotics with the T-Resilience Algorithm. *International Journal of Robotics Research*, 2013.

# Fast Damage Recovery in Robotics with the T-Resilience Algorithm

Sylvain Koos, Antoine Cully and Jean-Baptiste Mouret*

**Damage recovery is critical for autonomous robots that need to operate for a long time without assistance. Most current methods are complex and costly because they require anticipating each potential damage in order to have a contingency plan ready. As an alternative, we introduce the T-Resilience algorithm, a new algorithm that allows robots to quickly and autonomously discover compensatory behaviors in unanticipated situations. This algorithm equips the robot with a self-model and discovers new behaviors by learning to avoid those that perform differently in the self-model and in reality. Our algorithm thus does not identify the damaged parts but it implicitly searches for efficient behaviors that do not use them. We evaluate the T-Resilience algorithm on a hexapod robot that needs to adapt to leg removal, broken legs and motor failures; we compare it to stochastic local search, policy gradient and the self-modeling algorithm proposed by Bongard et al. The behavior of the robot is assessed onboard thanks to a RGB-D sensor and a SLAM algorithm. Using only 25 tests on the robot and an overall running time of 20 minutes, T-Resilience consistently leads to substantially better results than the other approaches.**



(a) Normal state. (b) Two legs ripped out. (c) One broken leg. (d) Two unpowered motors.

Figure 1: Examples of situations in which an autonomous robot needs to discover a qualitatively new behavior to pursue its mission: in each case, classic hexapod gaits cannot be used. The broken leg example (c) is a typical damage that is hard to diagnose by direct sensing (because no actuator or sensor is damaged).

## 1. Introduction

Autonomous robots are inherently complex machines that have to cope with a dynamic and often hostile environment. They face an even more demanding context when they operate for a long time without any assistance, whether when exploring remote places (Bellingham and Rajan, 2007) or, more prosaically, in a house without any robotics expert (Prassler and Kosuge, 2008). As famously pointed out by Corbato (2007), when designing such complex systems, "[we should not] wonder *if* some mishap may happen, but rather ask *what* one will do about it when it occurs". In autonomous robotics, this remark means that robots must be able to pursue their mission in situations that have not been anticipated by their designers. Legged robots clearly illustrate this need to handle the unexpected: to be as versatile as possible, they involve many moving parts, many actuators and many sensors (Kajita and Espiau, 2008); but they may be damaged in numerous different ways. These robots would therefore greatly benefit from being able to autonomously find a new behavior if some legs are ripped off, if a leg is broken or if one motor is inadvertently disconnected (Fig. 1).

Fault tolerance and resilience are classic topics in robotics and engineering. The most classic approaches combine intensive testing with redundancy of components (Visinsky et al., 1994; Koren and Krishna, 2007). These methods undoubtedly proved their usefulness in space, aeronautics and numerous complex systems, but they also are expensive to operate and to design. More importantly, they require the identification of the faulty subsystems and a procedure to bypass them, whereas both operations are difficult for many kinds of faults – for example mechanical failures. Another classic approach to fault tolerance is to employ robust controllers that can work in spite of damaged sensors or hardware inefficiencies (Goldberg and Chen, 2001; Caccavale and Villani, 2002; Qu et al., 2003; Lin and Chen, 2007). Such controllers usually do not require diagnosing the damage, but this advantage is tempered by the need to integrate the reaction to all faults in a single controller. Last, a robot can embed a few pre-designed behaviors to cope with anticipated potential failures (Görner and Hirzinger, 2010; Jakimovski and Maehle, 2010; Mostafa et al., 2010; Schleyer and Russell, 2010). For instance, if a hexapod robot detects that one of its legs is not reacting as expected, it can drop it and adapt the position of the other legs accordingly (Jakimovski and Maehle, 2010; Mostafa et al., 2010).

*Sylvain Koos, Antoine Cully and Jean-Baptiste Mouret are with the ISIR, Université Pierre et Marie Curie-Paris 6, CNRS UMR 7222, F-75252, Paris Cedex 05, France. Contact: mouret@isir.upmc.fr

An alternative and promising line of thought is to *let the robot learn on its own* the best behavior for the current situation. If the learning process is open enough, then the robot should be able to discover new compensatory behaviors in situations that have not been foreseen by its designers. Numerous learning systems have been experimented in robotics (for reviews, see Connell and Mahadevan (1993); Argall et al. (2009); Nguyen-Tuong and Peters (2011); Kober and Peters (2012)), with different levels of openness and various a priori constraints. Most of them primarily aim at automatically tuning controllers for complex robots (Kohl and Stone, 2004; Tedrake et al., 2005; Sproewitz et al., 2008; Hemker et al., 2009), but some of these systems have been explicitly tested in situations in which a robot needs to adapt itself to unexpected situations (Mahdavi and Bentley, 2003; Berenson et al., 2005; Bongard et al., 2006); the present work follows in their footsteps.

Finding the behavior that maximizes performance in the current situation is a *reinforcement learning problem* Sutton and Barto (1998), but classic reinforcement learning algorithms (e.g. TD-Learning, SARSA, ...) are designed for discrete state spaces (Sutton and Barto, 1998; Togelius et al., 2009). They are therefore hard to use when learning continuous behaviors such as locomotion patterns. Policy gradient algorithms (Kohl and Stone, 2004; Peters and Schaal, 2008; Peters, 2010) are reasonably fast learning algorithms that are better suited for robotics (authors typically report learning time of 20 minutes to a few hours), but they are essentially limited to a local search in the parameter space: they lack the openness of the search that is required to cope with truly unforeseen situations. Evolutionary Algorithms (EAs) (Deb, 2001; De Jong, 2006) can optimize reward functions in larger, more open search spaces (e.g. automatic design of neural networks, design of structures) (Grefenstette et al., 1999; Heidrich-Meisner and Igel, 2009; Togelius et al., 2009; Doncieux et al., 2011; Hornby et al., 2011; Whiteson, 2012), but this openness is counterbalanced by substantially longer learning time (according to the literature, 2 to 10 hours for simple robotic behaviors).

All policy gradient and evolutionary algorithms spend most of their running time in evaluating the quality of controllers by testing them on the target robot. Since, contrary to simulation, reality cannot be sped up, their running time can only be improved by finding strategies to evaluate fewer candidate solutions on the robot. In their "starfish robot" project, Bongard et al. (2006) designed a general approach for resilience that makes an important step in this direction. The algorithm of Bongard et al. is divided into two stages: (1) automatically building an internal simulation of the whole robot by observing the consequences of a few elementary actions (about 15 in the demonstrations of the paper) – this internal simulation of the whole body is called a *self-model*[1] (Metzinger, 2004, 2007; Vogeley et al., 1999; Bongard et al., 2006; Holland and Goodman, 2003; Hoffmann et al., 2010); (2) launching *in this simulation* an EA to find a new controller. In effect, this algorithm transfers most of the learning time to a computer simulation, which makes it increasingly faster when computers are improved (Moore, 1975).

Bongard's algorithm highlights how mixing a self-model with a learning algorithm can reduce the time required for a robot to adapt to an unforeseen situation. Nevertheless, it has a few important shortcomings. First, actions and models are undirected: the algorithm can "waste" a lot of time to improve parts of the self-model that are irrelevant for the task. Second, it is computationally expensive because it includes a full learning algorithm (the second stage, in simulation) and an expensive process to select each action that is tested on the robot. Third, there is often a "reality gap" between a behavior learned in simulation and the same behavior on the target robot (Jakobi et al., 1995; Zagal et al., 2004; Koos et al., 2012), but nothing is included in Bongard's algorithm to prevent such gap to happen: the controller learned in the simulation stage may not work well on the real robot, even if the self-model is accurate. Last, one can challenge the relevance of calling into question the full self-model each time an adaptation is required, for instance if an adaptation is only temporarily useful.

In the present paper we introduce a new resilience algorithm that overcomes these shortcomings while still performing most of the search in a simulation of the robot. Our algorithm works with any parametrized controller and it is especially efficient on modern, multi-core computers. More generally, it is designed for situations in which:

- behaviors optimized on the undamaged robot are not efficient anymore on the damaged robot (otherwise, adaptation is useless) and qualitatively new behavior is required (otherwise, local search algorithms should perform better);
- the robot can only rely on internal measurements of its state (truly autonomous robots do not have access to perfect, external sensing systems);
- some damages cannot be observed or measured directly (otherwise more explicit methods may be more efficient).

Our algorithm is inspired by the "transferability approach" (Koos et al., 2012; Mouret et al., 2012; Koos and Mouret, 2011), whose original purpose is to cross the "reality gap" that separates behaviors optimized in simulation to those observed on the target robot. The main proposition of this approach is to make the optimization algorithm aware of the limits of the simulation. To this end, a few controllers are transferred during the optimization and a regression algorithm (e.g. a SVM or a neural network) is used to approximate the function that maps behaviors in simulation to the difference of performance between simulation and reality. To use this approximated *transferability function*, the single-objective optimization problem is transformed into a multi-objective optimization in which both performance in simulation and transferability are maximized. This optimization is typically performed with a stochastic multi-objective optimization algorithm but other optimization algorithms are conceivable.

As this paper will show, the same concepts can be applied to design a fast adaptation algorithm for resilient robotics, leading to a new algorithm that we called "T-Resilience" (for Transferability-based resilience). If a damaged robot embeds a simulation of itself, then behaviors that rely on damaged parts will not be transferable: they will perform very differently in the self-model and in reality. During the adaptation process, the robot will thus create an approximated trans-

[1] Following the literature in psychology (Metzinger, 2004, 2007; Vogeley et al., 1999) and artificial intelligence (Bongard et al., 2006; Holland and Goodman, 2003), we define a self-model as a forward, internal model of the *whole body* that is accessible to introspection and instantiated in a model of the environment. In the present paper, we only consider a minimal model of the environment (a horizontal plane).

ferability function that classifies behaviors as "working as expected" and "not working as expected". Hence the robot will possess an "intuition" of the damages but it will not explicitly represent or identify them. By optimizing both the transferability and the performance, the algorithm will look for the most efficient behaviors among those that only use the reliable parts of the robots. The robot will thus be able to sustain a functioning behavior when damage occurs by learning to avoid behaviors that it is unable to achieve in the real world. Besides this damage recovery scenario, the T-Resilience algorithm opens a new class of adaptation algorithms that benefit from Moore's law by transferring most of the adaptation time from real experiments to simulations of a self-model.

We evaluate the T-Resilience algorithm on a hexapod robot that needs to adapt to leg removal, broken legs and motor failures; we compare it to stochastic local search (Hoos and Stützle, 2005), policy gradient (Kohl and Stone, 2004) and Bongard's algorithm (Bongard et al., 2006). The behavior on the real robot is assessed on-board thanks to a RGB-D sensor coupled with a state-of-the-art SLAM algorithm (Endres et al., 2012).

# 2. Learning for resilience

Discovering a new behavior after a damage is a particular case of *learning* a new behavior, a question that generates an abundant literature in artificial intelligence since its beginnings (Turing, 1950). We are here interested in reinforcement learning algorithms because we consider scenarios in which evaluating the performance of a behavior is possible but the optimal behavior is unknown. However, classic reinforcement learning algorithms are primarily designed for discrete states and discrete actions (Sutton and Barto, 1998; Peters, 2010), whereas autonomous robots have to solve many continuous problems (e.g. motor control). Two alternative families of methods are currently prevalent for continuous reinforcement learning in robotics (table 1): policy gradient methods and evolutionary algorithms. These two approaches both rely on optimization algorithms that directly optimize parameters of a controller by measuring the overall performance of the robot (Fig. 2); learning is thus here regarded as an optimization of these parameters.

## 2.1. Policy gradient methods

Policy gradient methods (Sutton et al., 2000; Peters and Schaal, 2008; Peters, 2010) use iterative stochastic optimization algorithms to find a local extremum of the reward function. The search starts with a controller that can be generated at random, designed by the user or inferred from a demonstration. The algorithm then iteratively modifies the parameters of the controller by estimating gradients in the control space and applying slight changes to the parameters.

Typical policy gradient methods iterate the following steps:

- generation of $N$ controllers in the neighborhood of the current vector of parameters (by variating one or multiple parameter values at once);
- estimation of the gradient of the reward function in the control space;

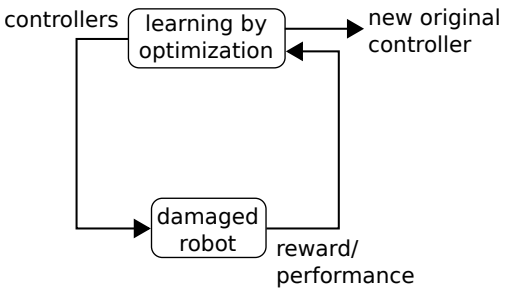



Figure 2: Principle of resilience processes based on policy gradient. Controllers are optimized by measuring rewards on the robot.

- modification of parameter values according to the gradient information.

These steps are iterated until a satisfying controller is found or until the process converges. Policy gradient algorithms essentially differ in the way gradient is estimated. The most simple way is the finite-difference method, which independently estimates the local gradient of the reward function for each parameter (Kohl and Stone, 2004; Tedrake et al., 2005): considering a given parameter, if higher (resp. lower) values lead to higher rewards on average on the $N$ controllers tested during the current iteration, the value of the parameter is increased (resp. decreased) for the next iteration. Such a simple method for estimating the gradient is especially efficient when parameters are mostly independent. Strong dependencies between the parameters often require more sophisticated estimation techniques.

Policy gradient algorithms have been successfully applied to locomotion tasks in the case of quadruped (Kimura et al., 2001; Kohl and Stone, 2004) and biped robots (Tedrake et al., 2005) but they typically require numerous evaluations on the robot, most of the times more than 1000 trials in a few hours (table 1). To make learning tractable, these examples all use carefully designed controllers with only a few degrees of freedom. They also typically start with well-chosen initial parameter values, making them efficient algorithms for imitation learning when these values are extracted from a demonstration by a human (Kober and Peters, 2010). Recent results on the locomotion of a quadruped robot suggest that using random initial controllers would likely require many additional experiments on the robot (Yosinski et al., 2011). Consistent results have been reported on biped locomotion with computer simulations using random initial controllers that make the robot fall (Nakamura et al., 2007) (about 10 hours of learning for 11 control parameters).

## 2.2. Evolutionary Algorithms

Evolutionary Algorithms (EAs) (Deb, 2001; De Jong, 2006) are another family of iterative stochastic optimization methods that search for the optima of function (Grefenstette et al., 1999; Heidrich-Meisner and Igel, 2009). They are less prone to local optima than policy gradient algorithms and they can optimize arbitrary structures (neural networks, fuzzy rules, vector of parameters, ...) (Doncieux et al., 2011; Hornby et al., 2011; Mouret and Doncieux, 2012; Whiteson, 2012).

While there exists many variants of EAs, the vast majority of them iterate the following steps:

- (first iteration only) random initialization of a population of candidate solutions;

Table 1: Typical examples of learning algorithms that have been used on legged robots.

| approach/article | starting beh. * | learning time | robot | DOFs† | param.‡ | reward |
|---|---|---|---|---|---|---|
| Policy Gradient Methods | | | | | | |
| Kimura et al. (2001) | no info. | 80 min. | quadruped | 8 | 72 | internal |
| Kohl and Stone (2004) | walking | 3 h | quadruped | 12 | 12 | external |
| Tedrake et al. (2005) | standing | 20 min. | bidepal | 2 | 46 | internal |
| Evolutionary Algorithm | | | | | | |
| Chernova and Veloso (2004) | random | 5 h | quadruped | 12 | 54 | external |
| Zykov et al. (2004) | random | 2 h | hexapod | 12 | 72 | external |
| Berenson et al. (2005) | random | 2 h | quadruped | 8 | 36 | external |
| Hornby et al. (2005) | non-falling | 25h | quadruped | 19 | 21 | internal |
| Mahdavi and Bentley (2006) | random | 10 h | snake | 12 | 1152 | external |
| Barfoot et al. (2006) | random | 10 h | hexapod | 12 | 135 | external |
| Yosinski et al. (2011) | random | 2 h | quadruped | 9 | 5 | external |
| Others | | | | | | |
| Weingarten et al. (2004) [1] | walking | $>$ 15 h | hexapod (Rhex-like) | 6 | 8 | external |
| Sproewitz et al. (2008) [2] | random | 60 min. | quadruped | 8 | 5 | external |
| Hemker et al. (2009) [3] | walking | 3-4 h | biped | 24 | 5 | external |
| Barfoot et al. (2006) [4] | random | 1h | hexapod | 12 | 135 | external |

*Behavior used to initialize the learning algorithm.
† DOFs: number of controlled degrees of freedom.
‡ param: number of learned control parameters.
[1] Nelder-Mead descent. [2] Powell method. [3] Design and Analysis of Computer Experiments. [4] Multi-agent reinforcement learning

- evaluation of the performance of each controller of the population (by testing the controller on the robot);
- ranking of controllers;
- selection and variation around the most efficient controllers to build a new population for the next iteration.

Learning experiments with EAs are reported to require many hundreds of trials on the robot and to last from two to tens of hours (table 1). EAs have been applied to quadruped robots (Hornby et al., 2005; Yosinski et al., 2011), hexapod robots (Zykov et al., 2004; Barfoot et al., 2006) and humanoids (Katić and Vukobratović, 2003; Palmer et al., 2009). EAs have also been used in a few studies dedicated to resilience, in particular on a snake-like robot with a damaged body (Mahdavi and Bentley, 2003) (about 600 evaluations/10 hours) and on a quadrupedal robot that breaks one of its leg (Berenson et al., 2005) (about 670 evaluations/2 hours).

Aside from these two main types of approaches, several authors proposed to use other black-box optimization algorithms: global methods like Nelder-Mead descent (Weingarten et al., 2004), local methods like Powell's method (Sproewitz et al., 2008) or surrogate-based optimization (Hemker et al., 2009). Published results are typically obtained with hundreds of evaluations on the robot, requiring several hours (table 1).

Regardless of the optimization technique, reward functions are, in most studies, evaluated with external tracking devices (table 1, last column). While this approach is useful when researchers aims at finding the most efficient controllers (e.g. Kohl and Stone (2004); Sproewitz et al. (2008); Hemker et al. (2009)), learning algorithms that target adaptation and resilience need to be robust to the inaccuracies and constraints of on-board measurements.

## 2.3. Resilience based on self-modeling

Instead of directly learning control parameters, Bongard et al. (2006) propose to improve the resilience of robots by equipping robots with a *self-model*. If a disagreement is detected between the self-model and observations, the proposed algorithm first infers the damages by chosing motor actions and measuring their consequences on the behavior of the robot; the algorithm then relies on the updated model of the robot to learn a new behavior. This approach has been successfully tested on a starfish-like quadrupedal robot (Bongard et al., 2006; Zykov, 2008). By adapting its self-model, the robot manages to discover a new walking gait after the loss of one of its legs.

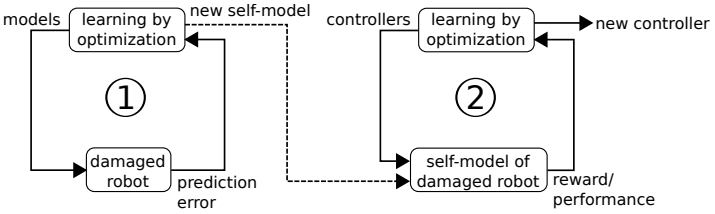



Figure 3: Principle of Bongard's algorithm. (1) A self-model is learned by testing a few actions on the damaged robot. (2) This self-model is next used as a simulation in which a new controller is optimized.

In Bongard's algorithm, the identification of the self-model is based on an active learning loop that is itself divided into an *action selection loop* and a *model selection loop* (Fig. 3). The action selection loop aims at selecting the action that will best distinguish the models of a population of candidate models. The model selection loop looks for the models that best predict the outcomes of the actions as measured on the robot. In the "starfish" experiment (Bongard et al., 2006), the following steps are repeated:

1.1. action selection (*exploration*):

- each of the 36 possible actions is tested on each of the 16 candidate models to observe the orientation of robot's body predicted by the model;
- the action for which models of the population dis-

agree at most is selected;

- this action is tested on the robot and the corresponding exact orientation of robot's body is recorded by an external camera;

1.2. model selection loop (*estimation*):

- a stochastic optimization algorithm (an EA) is used to optimize the population of models so that they accurately predict what was measured with the robot, for each tested action;
- if less than 15 actions have been performed, the action selection loop is started again.

Once the 15 actions have been performed, the best model found so far is used to learn a new behavior using an EA:

2. controller optimization (*exploitation*):

- a stochastic optimization algorithm (an EA) is used to optimize a population controllers so that they maximize forward displacement within the simulation of the self-model;
- the best controller found in the simulation is transferred to the robot, making it the new controller.

The population of models is initialized with the self-model that corresponds to the morphology of the undamaged robot. Since the overall process only requires 15 tests on the robot, its speed essentially depends on the performance of the employed computer. Significant computing times are nonetheless required for the optimization of the population of models.

In the results reported by Bongard et al. (2006), only half of the runs led to correct self-models. As Bongard's. approach implies identifying a full model of the robot, it would arguably require many more tests to converge in most cases to the right morphology. For comparison, results obtained by the same authors but in a simulated experiment required from 600 to 1500 tests to consistently identify the model (Bongard and Lipson, 2005). It should also be noted that these authors did not measure the orientation of robot's body with internal sensors, whereas noisy internal measurements could significantly impair the identification of the model. Other authors experimented with self-modeling process similar to the one of Bongard et al., but with a humanoid robot (Zagal et al., 2009). Preliminary results suggest that thousands of evaluations on the robot would be necessary to correctly identify 8 parameters of the global self-model. Alternative methods have been proposed to build self-models for robots and all of them require numerous tests, e.g. on a manipulator arm with about 400 real tests (Sturm et al., 2008) or on a hexapod robot with about 240 real tests (Parker, 2009). Overall, experimental costs for building self-models appear expensive in the context of resilience applications in both the number of tests on the real robot and in computing time.

Furthermore, controllers obtained by optimizing in a simulation – as does the algorithm proposed by Bongard et al. – often do no work as well on the real robot than in simulation (Koos et al., 2012; Zagal et al., 2004; Jakobi et al., 1995). In effect, this classic problem has been observed in the starfish experiments Bongard et al. (2006). In these experiments, it probably originates from the fact that the identified self-model cannot perfectly model every detail of the real world (in particular, slippage, friction and very dynamic behaviors).

### 2.4. Concluding thoughts

Based on this short survey of the literature, two main thoughts can be drawn:

1. Policy gradient methods and EAs can both be used to discover original behaviors on a damaged robot; nevertheless, when they don't start from already good initial controllers, they require a high number of real tests (at least a few hundred), which limits the speed of the resulting resilience process.

2. Methods based on self-modeling are promising because they transfer some of the learning time to a simulation; however building an accurate global model of the damaged robot requires many real tests; reality gap problems can also occur between the behavior learned with self-model and the real, damaged robot.

## 3. The T-Resilience algorithm

### 3.1. Concept and intuitions

Following Bongard et al., we equip our robot with a self-model. A direct consequence is that detecting the occurrence of a damage is facilitated: if the observed performance is significantly different from what the self-model predicts, then the robot needs to start a recovery process to find a better behavior. Nevertheless, contrary to Bongard et al., we propose that a damaged robot discovers new original behaviors *using the initial, hand-designed self-model*, that is without updating the self-model. Since we do not attempt to diagnose damages, the solved problem is potentially easier than the one solved by Bongard et al; we therefore expect our algorithm to perform faster. This speed increase can, however, comes at the price of slightly less efficient post-damage behaviors.

The model of the undamaged robot is obviously not accurate because it does not model the damages. Nonetheless, since damages can't radically change the overall morphology of the robot, this "undamaged" self-model can still be viewed as a reasonably accurate model of the damaged robot. Most of the degrees of freedom are indeed correctly positionned, the mass of components should not change much and the body plan is most probably not radically altered.

Imperfect simulators and models are an almost unavoidable issue when robotic controllers are first optimized in simulation then transferred to a real robot. The most affected field is probably evolutionary robotics because of the emphasis on opening the search space as much as possible: behaviors found within the simulation are often not anticipated by the designer of the simulator, therefore it's not surprising that they are often wrongly simulated. Researchers in evolutionary robotics explored three main ideas to cross this "reality gap": (1) automatically improving simulators (Bongard et al., 2006; Pretorius et al., 2012; Klaus et al., 2012), (2) trying to prevent optimized controllers from relying on the unreliable parts of the simulation (in particular, by adding noise) (Jakobi et al., 1995), and (3) model the difference between simulation and reality (Hartland and Bredeche, 2006; Koos et al., 2012).

Translated to resilient robotics, the first idea is equivalent to improving or adapting the self-model, with the aforementioned shortcomings (sections 1 and 2.3). The second idea

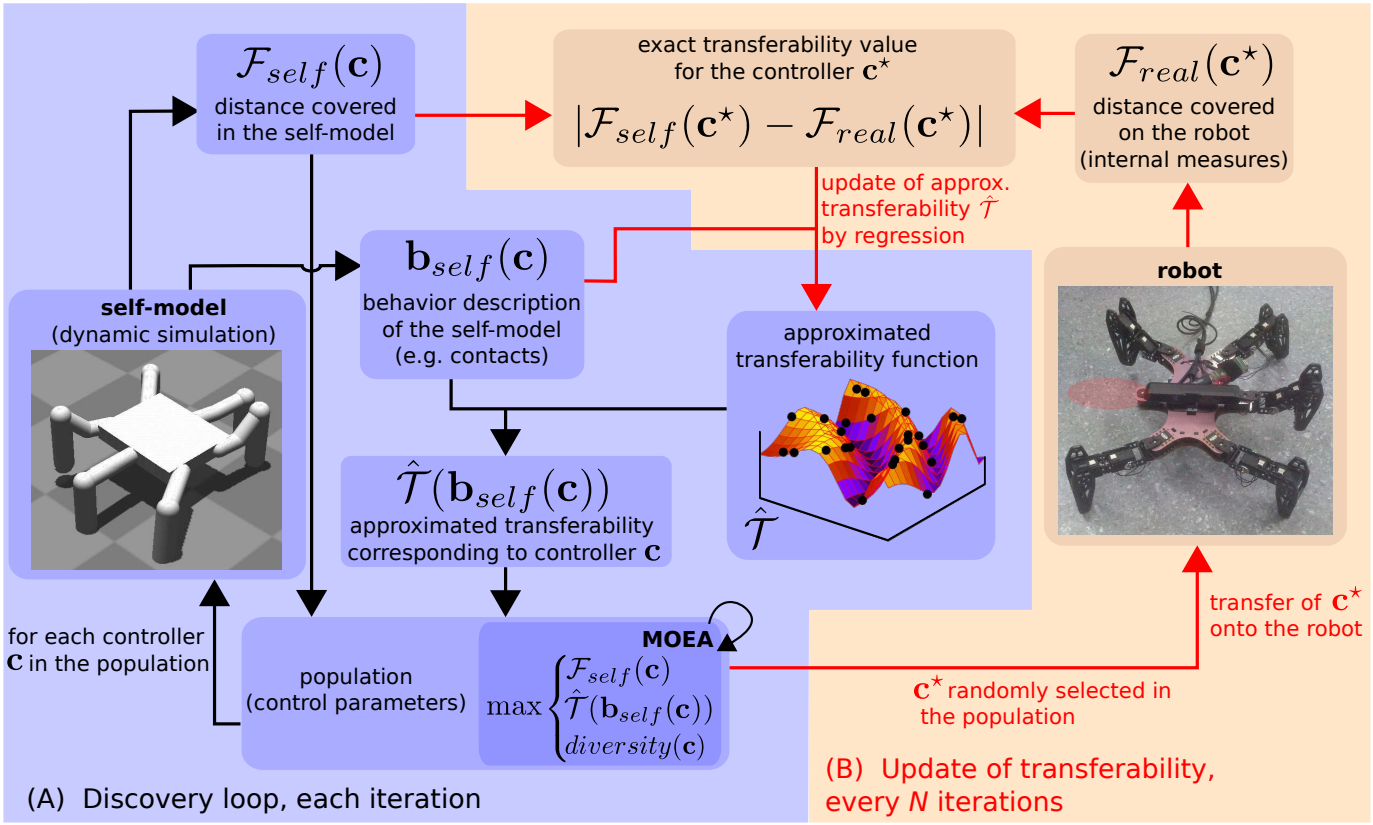



Figure 4: Schematic view of the T-Resilience algorithm (see algorithm 1 for an algorithmic view). (A) Discovery loop: each controller of the population is evaluated with the self-model. Its transferability score is approximated according to the current model $\hat{\mathcal{T}}$ of the exact transferability function $\mathcal{T}$. (B) Transferability update: every $N$ iterations, a controller of the population is randomly selected and transferred onto the real robot. The model of the transferability function is next updated with the data generated during the transfer.

corresponds to encouraging the robustness of controllers so that they can deal with an imperfect simulation. It could lead to improvements in resilient robotics but it requires that the designer anticipates most of the potential damages. The third idea is more interesting for resilient robotics because it acknowledges that simulations are never perfect and mixes reality and simulation during the optimization. Among the algorithms of this family, the recently-proposed transferability approach (Koos et al., 2012) explicitly searches for high-performing controllers that work similarly in both simulation and reality. It led to successful solutions for quadruped robot (2 parameters to optimize) and for a Khepera-like robot in a T-maze (weights of a feed-forward neural networks to optimize) (Koos et al., 2012; Koos and Mouret, 2011).

The main assumption of the transferability approach is that some transferable behaviors exist in the search space. Although formulated in the context of the reality gap, this assumption holds well in resilient robotics. For instance, if a hexapod robot breaks a leg, then gaits that do not critically rely on this leg should lead to similar trajectories in the self-model and on the damaged robot. Such gaits are numerous: those that make the simulated robot lift the broken leg so that it never hits the ground; those that make the robot walk on its "knees"; those that are robust to leg damages because they are closer to crawling than walking. Similar ideas can be found for most robots and for most mechanical and electrical damages, provided that there are different ways to achieve the mission. For example, any redundant robotic manipulator with a blocked joint should be able to follow a less efficient but working trajectory that does not use this joint.

The transferability approach captures the differences between the self-model and reality through the *transferability function* (Mouret et al., 2012; Koos et al., 2012):

**Definition 1 (transferability function)** *A* transferability function $\mathcal{T}$ *is a function that maps a vector* $\mathbf{b} \in \mathbb{R}^m$ *of* $m$ *solution descriptors (e.g. control parameters or behavior descriptors) to a* transferability score $\mathcal{T}(\mathbf{b})$ *that represents how well the simulation matches the reality for this solution (e.g. performance variation):*

$$\begin{array}{rccc} \mathcal{T}: & \mathbb{R}^m & \mapsto & \mathbb{R} \\ & \mathbf{b} & \mapsto & \mathcal{T}(\mathbf{b}) \end{array}$$

This function is usually not accessible because this would require to test every solution both in reality and in simulation (see Mouret et al. (2012) and Koos et al. (2012) for an example of exhaustive mapping). The transferability function can, however, be approximated with a regression algorithm (neural networks, support vector machines, etc.) by recording the behavior of a few controllers in reality and in simulation.

## 3.2. T-Resilience

To cross the reality gap, the transferability approach essentially proposes optimizing both the approximated transferability and the performance of controllers with a stochastic multi-objective optimization algorithm. This approach can be adapted to make a robot resilient by seeing the original, "un-damaged" self-model as an inaccurate simulation of

**Algorithm 1** T-Resilience ($T$ real tests)

$pop \leftarrow \{c^1, c^2, \ldots, c^S\}$ (randomly generated)
$data \leftarrow \emptyset$

**for** $i = 1 \rightarrow T$ **do**

| | |
|---|---|
| random selection of $c^*$ in $pop$<br>computation of $\mathbf{b}_{self}(c^*)$, vector of $m$ values describing $c^*$ in the self-model<br>transfer of $c^*$ on the robot<br>estimation of performance $\mathcal{F}_{real}(c^*)$ using internal measurements<br>estimation of transferability score $\mathcal{T}(\mathbf{b}_{self}(c^*)) = \|\mathcal{F}_{self}(c^*) - \mathcal{F}_{real}(c^*)\|$<br>$data \leftarrow data \cup \{[\mathbf{b}_{self}(c^*), \mathcal{T}(\mathbf{b}_{self}(c^*))]\}$<br>learning of new approximated transferability function $\hat{\mathcal{T}}$, based on $data$ | (B) Updating approx. transferability function |
| $N$ iterations of MOEA on $pop$ by maximizing $\mathcal{F}_{self}(c)$, $\hat{\mathcal{T}}(\mathbf{b}_{self}(c))$, $diversity(c)$ | (A) Discovery loop |

**end for**

selection of the new controller

the damaged robot, and if the robot only uses internal measurements to evaluate the discrepancies between predictions of the self-model and measures on the real robot. Resilient robotics is thus a related, yet new application of the transferability concept. We call this new approach to resilient robotics "T-Resilience" (for Transferability-based Resilience).

**Algorithm.** The T-Resilience algorithm relies on three main principles (Fig. 4 and Algorithm 1):

- the self-model of the robot is not updated;
- the approximated transferability function is learned "on the fly" thanks to a few periodic tests conducted on the robot and a regression algorithm;
- three objectives are optimized simultaneously:

$$\text{maximize} \begin{cases} \mathcal{F}_{self}(\mathbf{c}) \\ \hat{\mathcal{T}}(\mathbf{b}_{self}(\mathbf{c})) \\ diversity(\mathbf{c}) \end{cases}$$

where $\mathcal{F}_{self}(\mathbf{c})$ denotes the performance of the candidate solution $\mathbf{c}$ that is predicted by the self-model (e.g. the forward displacement in the simulation); $\mathbf{b}_{self}(\mathbf{c})$ denotes the behavior descriptor of $\mathbf{c}$, extracted by recording the behavior of $\mathbf{c}$ in the self-model; $\hat{\mathcal{T}}(\mathbf{b}_{self}(\mathbf{c}))$ denotes the approximated transferability function between the self-model and the damaged robot, which is separately learned using a regression algorithm; and $diversity(\mathbf{c})$ is a application-dependent helper-objective that helps the optimization algorithm to mitigate premature convergence (Toffolo and Benini (2003); Mouret and Doncieux (2012)).

Evaluating these three objectives for a particular controller does not require any real test: the behavior of each controller and the corresponding performance are predicted by the self-model; the approximated transferability value is computed thanks to the regression model of the transferability function. *The update of the approximated transferability function is therefore the only step of the algorithm that requires a real test on the robot.* Since this update is only performed every $N$ iterations of the optimization algorithm, only a handful of tests on the real robot have to be done.

At a given iteration, the T-Resilience algorithm does not need to predict the transferability of the whole search space, it only needs these values for the candidate solutions of the current population. Since the population, on average, moves towards better solutions, the algorithm has to periodically update the approximation of the transferability function. To make this update simple and unbiased, we chose to select the solution to be tested on the robot by picking a random individual from the population. We experimented with other selection schemes in preliminary experiments, but we did not observe any significant improvement.

Three choices depend on the application:

- the performance measure $\mathcal{F}_{self}$ (i.e. the reward function);
- the diversity measure;
- the regression technique used to learn the transferability function and, in particular, the inputs and outputs of this function.

We will discuss and describe each of these choices for our resilient hexapod robot in section 4.

**Optimization algorithm.** Recent research in stochastic optimization proposed numerous algorithms to simultaneously optimize several objectives (Deb, 2001); most of them are based on the concept of Pareto dominance, defined as follows:

**Definition 2 (Pareto dominance)** *A solution $p^*$ is said to dominate another solution $p$, if both conditions 1 and 2 are true:*

1. *the solution $p^*$ is not worse than $p$ with respect to all objectives;*
2. *the solution $p^*$ is strictly better than $p$ with respect to at least one objective.*

The non-dominated set of the entire feasible search space is the globally Pareto-optimal set (Pareto front). It represents the set of optimal *trade-offs*, that is solutions that cannot be improved with respect to one objective without decreasing their score with respect to another one.

Pareto-based multi-objective optimization algorithms aim at finding the best approximation of the Pareto front, both in terms of distance to the Pareto front and of uniformity of its sampling. This Pareto front is found using only one execution of the algorithm and the choice of the final solution is left to another algorithm (or to the researcher). Whereas classic approaches to multi-objective optimization aggregate objectives (e.g. with a weighted sum) then use a single-objective

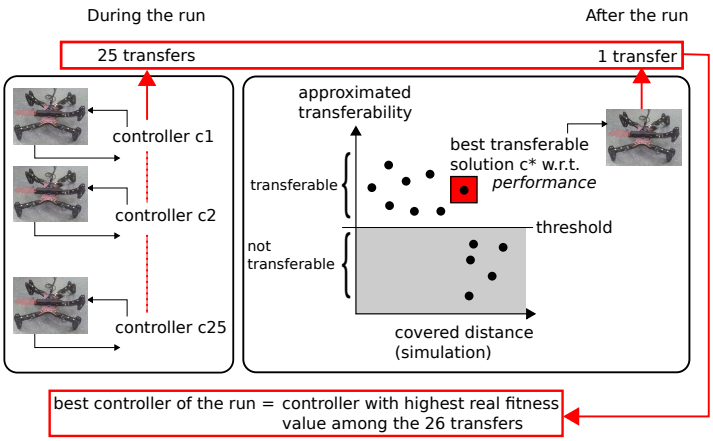



Figure 5: Choice of the final solution at the end of the T-Resilience algorithm.

optimization algorithm, multi-objective optimization algorithms do not require tuning the relative importance of each objective.

Current stochastic algorithms for multi-objective optimization are mostly based on EAs, leading to Multi-Objective Evolutionary Algorithms (MOEA). Like most EAs, they are intrinsically parallel (Cantu-Paz, 2000), making them especially efficient on modern multi-core computers, GPUs and clusters (Mouret and Doncieux, 2010). In the T-Resilience algorithm, we rely on NSGA-II (Deb et al., 2002; Deb, 2001), one of the most widely used multi-objective optimization algorithm (appendix B); however, any Pareto-based multi-objective algorithm can replace this specific EA in the T-Resilience algorithm.

At the end of the optimization algorithm, the MOEA discards diversity values and returns a set of non-dominated solutions based on performance and transferability. We then need to choose the final controller. Let us define the "transferable non-dominated set" as the set of non-dominated solutions whose transferability values are greater than a user-defined threshold. To determine the best solution of a run, the solution of the transferable non-dominated set with the highest performance in simulation is transferred onto the robot and its performance in reality is assessed. The final solution of the run is the controller that leads to the highest performance on the robot among all the transferred controllers (Fig. 5).

# 4. Experimental validation

## 4.1. Robot and parametrized controller

The robot is a hexapod with 18 Degrees of Freedom (DOF), 3 for each leg (Fig. 6(a,c)). Each DOF is actuated by position-controlled servos (6 AX-12 and 12 MX-28 Dynamixel actuators, designed by Robotis). 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 6 c.

A RGB-D camera (Asus Xtion) is screwed on top of the robot. It is used to estimate the forward displacement of the robot thanks to a RGB-D SLAM algorithm (Endres et al., 2012)[2] from the ROS framework (Quigley et al., 2009)[3].

The movement of each DOF is governed by a periodic function that computes its angular position as a function $\gamma$ of time $t$, amplitude $\alpha$ and phase $\phi$ (Fig. 6, d):

$$\gamma(t,\ \alpha,\ \phi) = \alpha \cdot \tanh\left(4 \cdot \sin\left(2 \cdot \pi \cdot (t + \phi)\right)\right) \quad (1)$$

where $\alpha$ and $\phi$ are the parameters that define the amplitude of the movement and the phase shift of $\gamma$, respectively. 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 control signal of the third servo is the opposite of the second one. 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;
- $-\gamma(t,\ \alpha_2^i,\ \phi_2^i)$ for DOFs 3.

This controller makes the robot equivalent to a 12 DOFs system, even if 18 motors are controlled.

There are $4$ parameters for each leg ($\alpha_1^i$, $\alpha_2^i$, $\phi_1^i$, $\phi_2^i$), therefore each controller is fully described by $24$ parameters. By varying these $24$ parameters, numerous gaits are possible, from purely quadruped gaits to classic tripod gaits.

This controller is designed to be as simple as possible so that we can show the performance of the T-Resilience algorithm in a straightforward setup. Nevertheless, the T-Resilience algorithm does not put any constraint on the type of controllers and many other controllers are conceivable (e.g. bio-inspired central pattern generators like Sproewitz et al. (2008) or evolved neural networks like in (Yosinski et al., 2011; Clune et al., 2011)).

## 4.2. Reference controller

A classic tripod gait (Wilson, 1966; Saranli et al., 2001; Schmitz et al., 2001; Ding et al., 2010; Steingrube et al., 2010) is used as a reference point. This reference gait considers two tripods: legs 0, 2, 4 and legs 1, 3, 5 (see Figure 6 for numbering). It is designed to always keep the robot balanced on at least one of these tripods. The walking gait is achieved by lifting one tripod, while the other pushes the robot forward (by shifting itself backward). The lifted tripod is then placed forward in order to repeat the cycle by inverting the tripods. This gait is static, fast and similar to insect gaits (Wilson, 1966; Delcomyn, 1971). The parameters of this reference controller are available in appendix C.

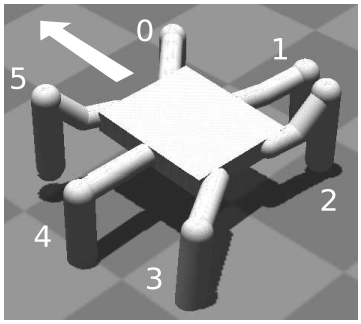
## 4.3. Implementation choices for T-Resilience

**Performance function.** The mission of our hexapod robot is to go forward as fast as possible, regardless of its current state and of any sustained damages. The performance function to be optimized is the forward displacement of the robot predicted by its self-model. Such a high-level function does not constrain the features of the optimized behaviors, so that the search remains as open as possible, possibly leading to original gaits (Nelson et al., 2009):

[2] We downloaded our implementation from: `http://www.ros.org/wiki/rgbdslam`

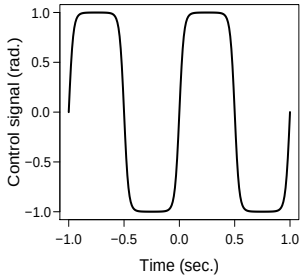
[3] `http://www.ros.org`

(a) Hexapod robot.




(b) Self-model.

(c) Kinematic scheme.




(d) Control function.

Figure 6: (a) The 18-DOF hexapod robot is equipped with a RGB-D camera (RGB camera with a depth sensor). (b) Snapshot of the simulation used as a self-model by the robot which occurs in an ODE-based physics simulator. The robot lies on a horizontal plane and contacts are simulated as well. (c) Kinematic scheme of the robot. (d) Control function $\gamma(t,\ \alpha,\ \phi)$ with $\alpha = 1$ and $\phi = 0$.

$$\mathcal{F}_{self}(\mathbf{c}) = p_x^{t=E,SELF}(\mathbf{c}) - p_x^{t=0,SELF}(\mathbf{c}) \quad (2)$$

where $p_x^{t=0,SELF}(\mathbf{c})$ denotes the x-position of the robot's center at the beginning of the simulation when the parameters $\mathbf{c}$ are used and $p_x^{t=E,SELF}(\mathbf{c})$ its x-position the end of the simulation.

Because each trial lasts only a few seconds, this performance function does not strongly penalize gaits that do not lead to straight trajectories. Using longer experiments would penalize these trajectories more, but it would increase the total experimental time too much to perform comparisons between approaches. Other performance functions are possible and will be tested in future work.

**Diversity function.** The diversity score of each individual is the average Euclidean distance to all the other candidate solutions of the current population. Such a parameter-based diversity objective enhances the exploration of the control space by the population (Toffolo and Benini, 2003; Mouret and Doncieux, 2012) and allows the algorithm to avoid many local optima. This diversity objective is straightforward to implement and does not depend on the task.

$$diversity(\mathbf{c}) = \frac{1}{N} \sum_{y \in P_n} \sqrt{\sum_{j=1}^{24} (\mathbf{c}_j - \mathbf{y}_j)^2} \quad (3)$$

where $P_n$ is the population at generation $n$, $N$ the size of $P$ and $\mathbf{c}_j$ the $j^{th}$ parameter of the candidate solution $\mathbf{c}$. Other diversity measures (e.g. behavioral measures, like in (Mouret and Doncieux, 2012)) led to similar results in preliminary experiments.

**Regression model.** When a controller $\mathbf{c}$ is tested on the real robot, the corresponding exact transferability score $\mathcal{T}$ is computed as the absolute difference between the forward performance predicted by the self-model and the performance estimated on the robot based on the SLAM algorithm.

$$\mathcal{T}(\mathbf{c}) = \left| p_{t=E}^{SELF}(\mathbf{c}) - p_{t=0}^{REAL}(\mathbf{c}) \right| \quad (4)$$

The transferability function is approximated by training a SVM model $\hat{\mathcal{T}}$ using the $\nu$-Support Vector Regression algorithm with linear kernels implemented in the library *libsvm*[4] (Chang and Lin, 2011) (learning parameters are set to default values).

$$\hat{\mathcal{T}}(\mathbf{b}_{self}(\mathbf{c})) = \text{SVM}(b_{t=0}^{(1)}, \cdots, b_{t=E}^{(1)}, \cdots, b_{t=0}^{(6)}, \cdots, b_{t=E}^{(6)}) \quad (5)$$

where $E$ is the number of time-steps of the control function (equation 1) and:

$$b_t^{(n)} = \begin{cases} 1 & \text{if leg } n \text{ touches the ground at that time-step} \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

We chose to describe gaits using contacts[5], because it is a classic representation of robotic and animal gaits (e.g. Delcomyn (1971)). On the real robots, we deduces the contacts by measuring the torque applied by each servo.

We chose SVMs to approximate the transferability score because of the high number of inputs of the model and because there are many available implementations. Contrary to other classic regression models (neural networks, Kriging, ...), SVMs are indeed not critically dependent on the size of the input space (Smola and Vapnik, 1997; Smola and Schölkopf, 2004). They also provide fast learning and fast prediction when large input spaces are used.

**Self-model.** The self-model of the robot is a dynamic simulation of the undamaged six-legged robot in Open Dynamics Engine (ODE)[6] on a flat ground (Fig. 6b).

**Main parameters.** For each experiment, a population of 100 controllers is optimized for 1000 generations. Every 40 generations, a controller is randomly selected in the population and transferred on the robot, that is we use 25 real tests on the robot in a run. Each test takes place as follows:

[4] `http://www.csie.ntu.edu.tw/~cjlin/libsvm`

[5] When choosing the input of a predictor, there is a large difference between using the control parameters and using high-level descriptors of the behavior (Mouret and Doncieux, 2012). Intuitively, most humans can predict that a behavior will work on a real robot by watching a simulation, but their task is much harder if they can only see the parameters. More technically, predicting features of a complex dynamical system usually requires simulating it. By starting with the output of a simulator, the predictor avoids the need to re-invent physical simulation and can focus on discrimination.

[6] Open Dynamics Engine: `http://www.ode.org`

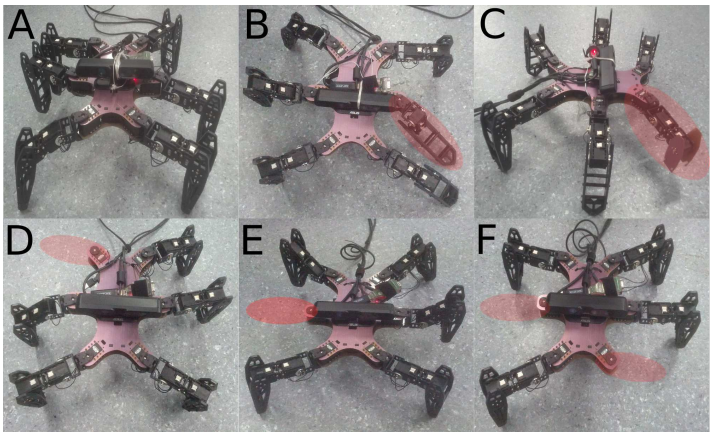



Figure 7: Test cases considered in our experiments. (A) The hexapod robot is not damaged. (B) The left middle leg is no longer powered. (C) The terminal part of the front right leg is shortened by half. (D) The right hind leg is lost. (E) The middle right leg is lost. (F) Both the middle right leg and the front left leg are lost.

- the selected controller is transferred and evaluated for 3 seconds on the robot while the RGB-D camera records both color and depth images at 10 Hz;
- a SLAM algorithm estimates the forward displacement of the robot based on the data of the camera;
- the estimate of the forward displacement is provided to the main algorithm.

At each generation, each parameter of each selected candidate solution has a $10\%$ chance of being incremented or decremented, with both options equally likely; five values are available for each $\varphi$ $(0, 0.25, 0.5, 0.75, 1)$ and for each $\alpha$ $(0, 0.25, 0.5, 0.75, 1)$.

To select the final solution, we fixed the transferability threshold at 0.1 meter.

## 4.4. Test cases and compared algorithms

To assess the ability of T-Resilience to cope with many different failures, we consider the six following test cases (Fig. 7):

- A. the hexapod robot is not damaged;
- B. the left middle leg is no longer powered;
- C. the terminal part of the front right leg is shortened by half;
- D. the right hind leg is lost;
- E. the middle right leg is lost;
- F. both the middle right leg and the front left leg are lost.

We compare the The T-Resilience algorithm to three representative algorithms from the literature (see appendix D for the exact implementations of each algorithm and appendix E for the validation of the implementations):

- a stochastic local search (Hoos and Stützle, 2005), because of its simplicity;
- a policy gradient method inspired from Kohl and Stone (2004), because this algorithm has been successfully applied to learn quadruped locomotion;
- a self-modeling process inspired from Bongard et al. (2006).

To make the comparisons as fair as possible, we designed our experiments to compare algorithms after the same amount of running time or after the same number of real tests (see appendix F for their median durations and their median numbers of real tests). In all the test cases, the T-Resilience algorithm required about 19 minutes and 25 tests on the robot (1000 generations of 100 individuals). Consequently, two key values are recorded for each algorithm (see Appendix D for exact procedures):

- the performance of the best controller obtained after about 25 real tests[7];
- the performance of the best controller obtained after about 19 minutes.

The experiments for the four first cases (A, B, C and D) showed that only the stochastic local search is competitive with the T-Resilience. To keep experimental time reasonable, we therefore chose to only compare T-Resilience with the local search algorithm for the two last failures (E and F).

Preliminary experiments with each algorithm showed that initializing them with the parameters of the reference controller did not improve their performance. We interpret these preliminary experiments as indicating that the robot needs to use a qualitatively different gait, which requires substantial changes in the parameters. This observation is consistent with the gaits we tried to design for the damaged robot. As a consequence, we chose to initialize each of the compared algorithms with random parameters instead of initializing them with the parameters of the reference controller. By thus starting with random parameters, we do not rely on any a priori about the gaits for the damaged robot: we start with the assumption that anything could have happened.

We replicate each experiment 5 times to obtain statistics. Overall, this comparison requires the evaluation of about 4000 different controllers on the real robot.

We use 4 Intel(R) Xeon(R) CPU E31230 3.20GHz, each of them including 4 cores. Each algorithm is programmed in the Sferes$_{v2}$ framework (Mouret and Doncieux, 2010) and the source-code is available as extension 10. The MOEA used in Bongard's algorithm and in the T-Resilience algorithm is distributed on 16 cores using MPI.

Final performance values are recorded with a CODA cx1 motion capture system (Charnwood Dynamics Ltd, UK) so that reported results do not depend on inaccuracies of the internal measurements. However, all the tested algorithms have only access to the internal measurements.

## 4.5. Using predefined controllers

One of the main strength of the T-Resilience algorithm is that it does not rely on any assumption about the failure. Nevertheless, the number of potential failures on a hexapod robot may appear quite limited: either one leg is unusable or two legs are unusable (if more legs are broken, then the robot is most probably unable to walk). For each of these 21 potential failures, a specific gait can be designed in the lab, and these behaviors may be sufficient to cope with any failure. The recovery process then consists in testing 22 controllers (these 21 controllers and the reference one) and using the best performing one.

To show that our algorithm is able to handle more diverse situations than this simple process, we consider an experi-

[7] Depending on the algorithm, it is sometimes impossible to perform exactly 25 tests (for instance, if two tests are performed for each iteration).

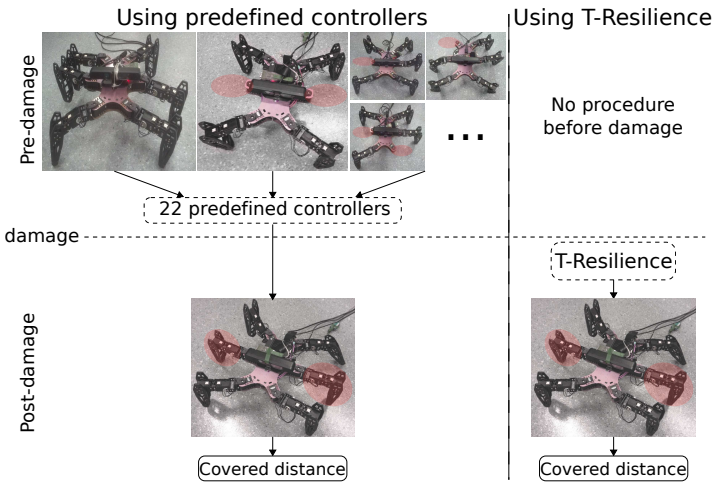



Figure 8: Using 22 predefined controllers is an alternative to learning new gaits. When the robot undergoes damages, all of these controllers are tested on the robot and the best performing one is selected. In the proposed experiment, we compare the efficiency of using these predefined controllers (left) with the direct application of the T-Resilience algorithm (right) in the following case: the two middle legs of the robot are blocked in their initial position.

ment in which the two middle legs of the robot are blocked in their initial position (Figure 8). This failure can easily happen when a wire is deficient in a data bus. Although this failure is not specifically anticipated by the described recovery process, the controller designed for a robot without the two middle legs might perform well. The 20 other controllers are most probably irrelevant to cope with this specific damage. Since designing 20 high-performing controllers is a very time-consuming process that is out of the scope of the present article, we ignore them in this experiment.

We use an evolutionary learning algorithm to synthesize the controller for the case of the two middle legs lost, so that this controller is not biased against the tested failure. To obtain controllers that work on the robot, we exploit the transferability approach (Koos et al., 2012; Mouret et al., 2012) with the accurate self-model (i.e. without the two middle legs). Because this optimization process is stochastic, we replicate it 10 times and obtain 10 optimized walking controllers. We then evaluate each optimized controller on the robot with the two blocked legs, and we record the covered distance. We compare the result to the direct application of the T-Resilience algorithm on the robot with two blocked middle legs (Figure 8). The T-Resilience experiments are replicated 10 times, each one with 25 real tests on the robot.

For completness, the reference controller is also tested on the robot with two blocked middle legs.

# 5. Results

## 5.1. Reference controller

Table 2 reports the performances of the reference controller for each tested failure, measured with both the CODA scanner and the on-board SLAM algorithm. At best, the damaged robot covered $35\%$ of the distance covered by the undamaged robot ($0.78$ m with the undamaged robot, at best $0.26$ m after a failure). In cases B, C and E, the robot also performs about a quarter turn (Figure 9 (a), (b) and (e)); in case D, it falls over; in case F, it alternates forward locomotion and backward locomotion (figure 9 (f)). Videos of these behaviors are available in appendix A.

| Test cases | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| Perf. (CODA) | 0.78 | 0.26 | 0.25 | 0.00 | 0.15 | 0.10 |
| Perf. (SLAM) | 0.75 | 0.17 | 0.26 | 0.00 | 0.04 | 0.16 |

Table 2: Performances in meters obtained on the robot with the reference gait in all the considered test cases. Each test lasts 3 seconds. The CODA line corresponds to the distance covered by the robot according to the external motion capture system. The SLAM line corresponds to the performance of the same behaviors but reported by the SLAM algorithm. When internal measures are used (SLAM line), the robot can easily detects that a damage occurred because the difference in performance is very significant (column A versus the other columns).

This performance loss of the reference controller clearly shows that an adaptation algorithm is required to allow the robot to pursue its mission. Although not perfect, the distances reported by the on-board RGB-D SLAM are sufficiently accurate to easily detect when the adaptation algorithm must be launched.

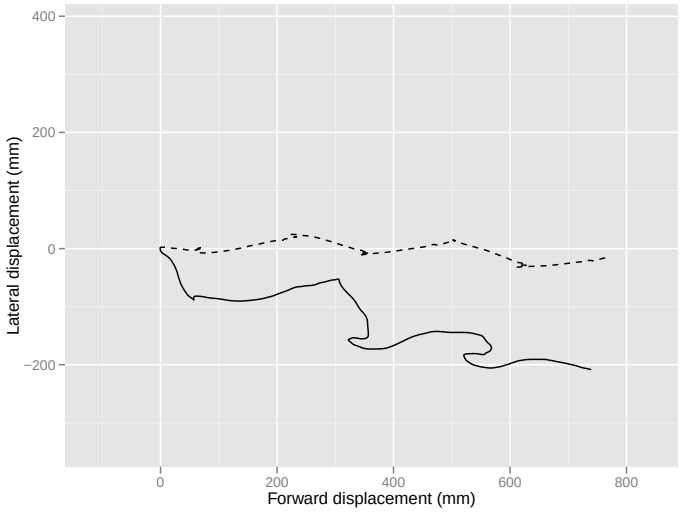
## 5.2. Comparison of performances

Fig. 10 shows the performance obtained for all test cases and all the investigated algorithms. Table 3 reports the improvements between median performance values. P-values are computed with the Wilcoxon rank-sum tests (appendix G). The horizontal lines in Figure 10 show the efficiency of the reference gait in each case.

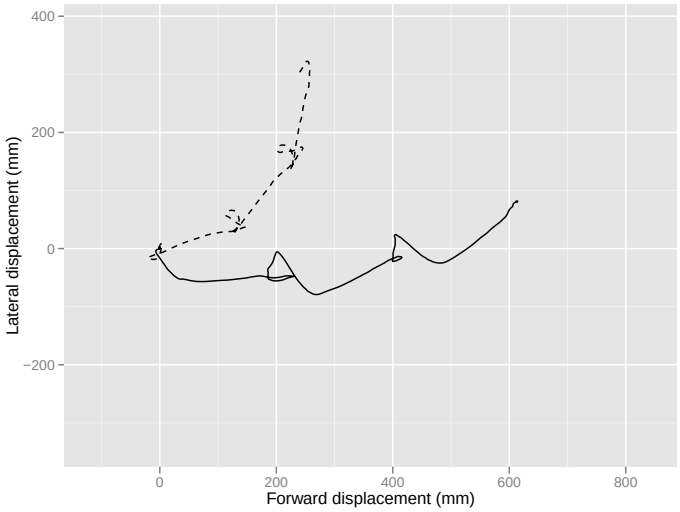
The trajectories corresponding to controllers with median performance values obtained with the T-Resilience are depicted on figure 9. Videos of the typical behaviors obtained with the T-Resilience on every test case are available in extension (Extensions 1 to 9).

**Performance with the undamaged robot (case A).** When the robot is not damaged, the T-Resilience algorithm discovered controllers with the same level of performance than the reference hexapod gait (p-value $= 1$). The obtained controllers are from 2.5 to 19 times more efficient than controllers obtained with other algorithms (Table 3).

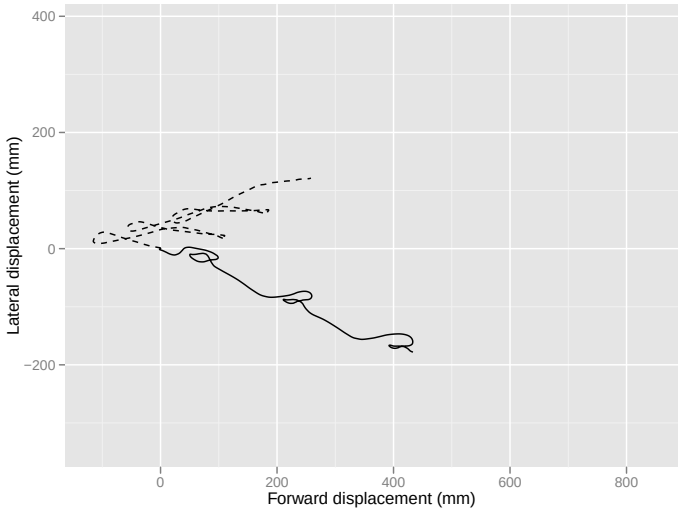
The poor performance of the other algorithms may appear surprising at first sight. Local search is mostly impaired by the very low number of tests that are allowed on the robot, as suggested by the better performance of the "time" variant (20 minutes / 50 tests) versus the "tests" variant (10 minutes / 25 tests). Surprisingly, we did not observe any significant difference when we initialized the control parameters with those of the reference controller (data not shown). The policy gradient method suffers even more than local search from the low number of tests because a lot of tests are required to estimate the gradient. As a consequence, we were able to perform only 2 to 4 iterations of the algorithm. Overall, these results are consistent with those of the literature because previous experiments used longer experiments and often simpler systems. Similar observations have been reported previously by other authors (Yosinski et al., 2011).



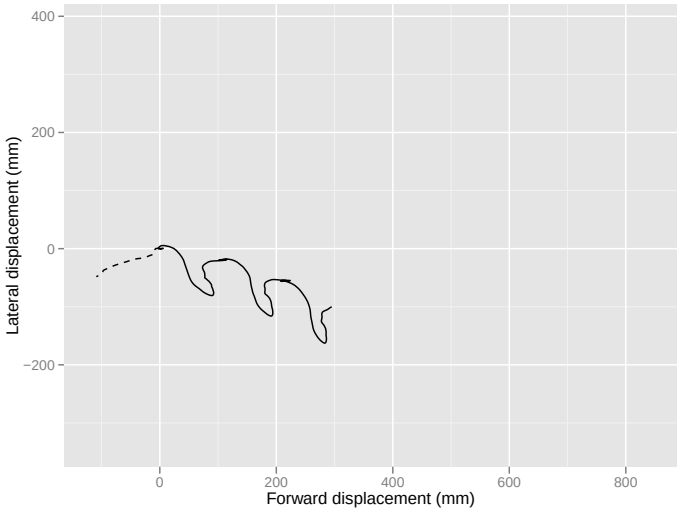

(a) Undamaged hexapod robot (case A).



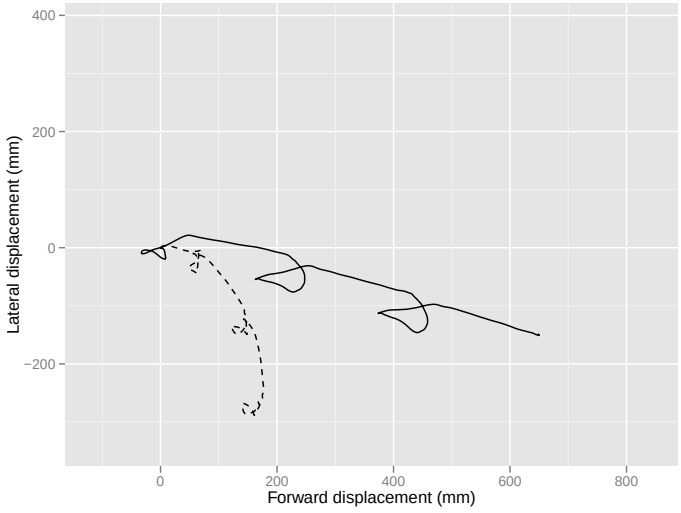

(b) Middle left leg not powered (case B).



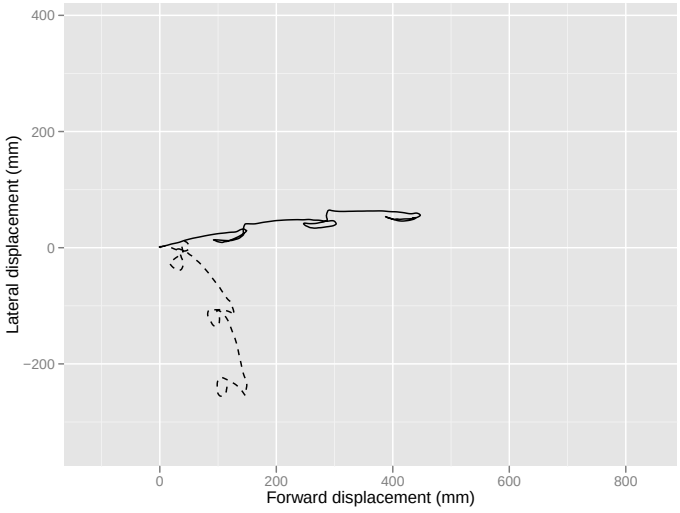

(c) Front right leg shortened by half (case C).




(d) Hind right leg lost (case D).

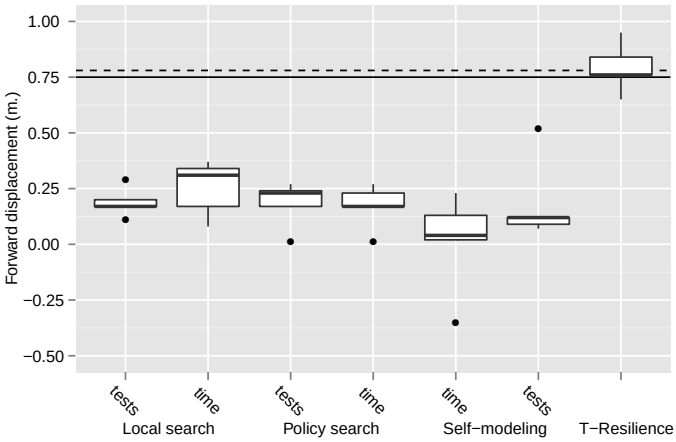



(e) Middle right leg lost (case E).

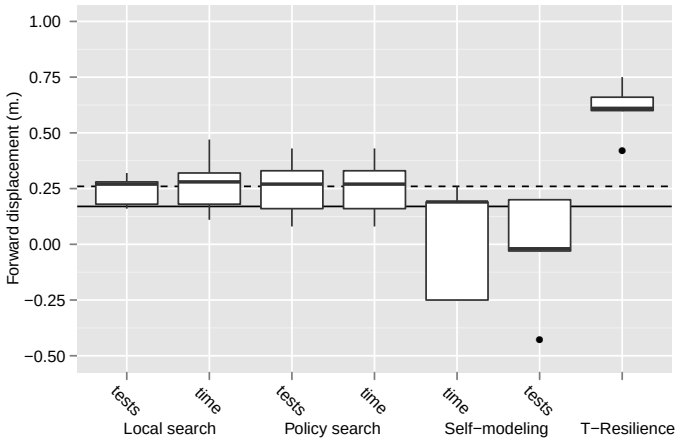



(f) Middle right leg and front left leg lost (case F).

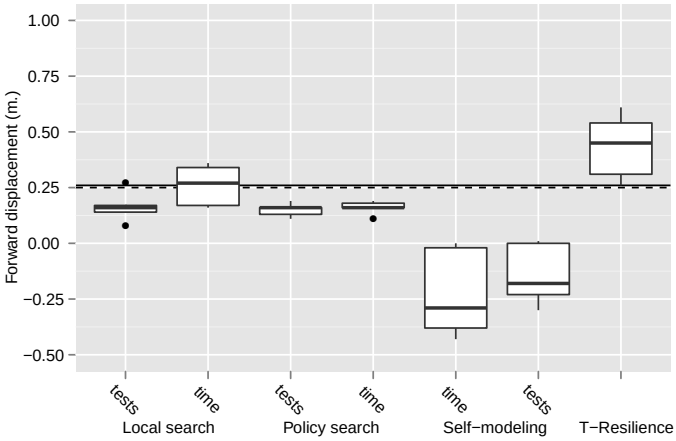
Figure 9: Typical trajectories (median performance) observed in every test case. Dashed line: reference gait. Solid line: controller with median performance value found by the T-Resilience algorithm. The poor performance of the reference controllers after any of the damages shows that adaptation is required in these situations. The trajectories obtained with the T-Resilience algorithm are not perfectly straight because our objective function does not explicitly reward straightness (see sections 4.3 and 5.2).



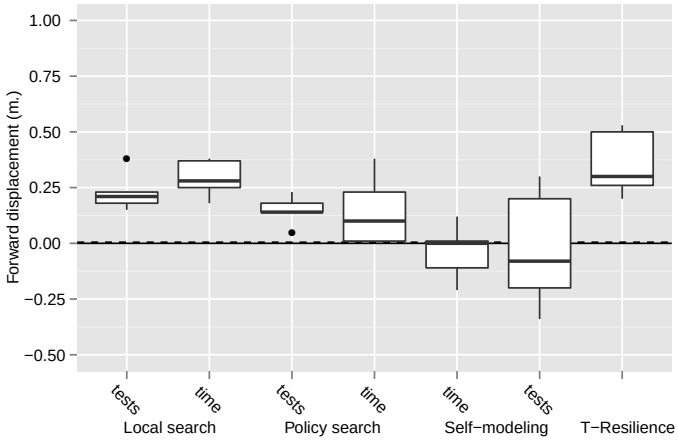

(a) Undamaged hexapod robot (case A).



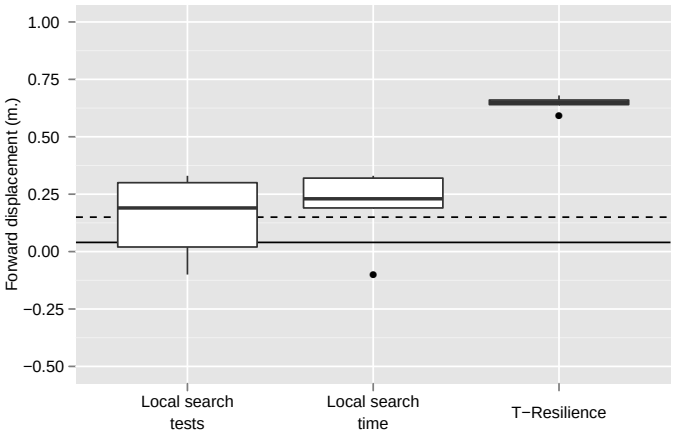

(b) Middle left leg not powered (case B).



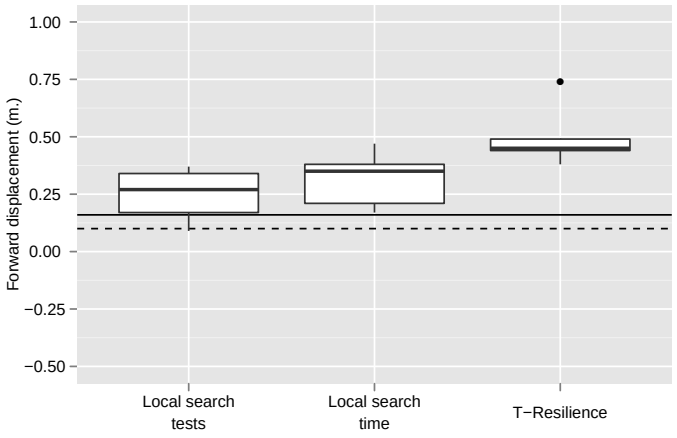

(c) Front right leg shortened by half (case C).




(d) Hind right leg lost (case D).




(e) Middle right leg lost (case E).




(f) Middle right leg and front left leg lost (case F).

Figure 10: Performances obtained in each test cases (distance covered in 3 seconds). On each box, the central mark is the median, the edges of the box are the lower hinge (defined as the 25th percentile) and the upper hinge (the 75th percentile). The whiskers extend to the most extreme data point which is no more than 1.5 times the length of the box away from the box. Each algorithm has been run 5 times and distances are measured using the external motion capture system. Except for the T-Resilience, the performance of the controllers found after about 25 transfers ($tests$) and after about 20 minutes ($time$) are depicted (all T-Resilience experiments last about 20 minutes and use 25 transfers). The horizontal lines denote the performances of the reference gait, according to the CODA scanner (dashed line) and according to the SLAM algorithm (solid line).

| | Local search | | Policy search | | Self-modeling | | reference gait |
|---|---|---|---|---|---|---|---|
| | tests | time | tests | time | time | tests | |
| A | 4.5 | 2.5 | 3.3 | 4.5 | 6.3 | 19.0 | 1.0 |
| B | 2.3 | 2.2 | 2.3 | 2.3 | +++ | 3.2 | 2.3 |
| C | 2.8 | 1.7 | 2.8 | 2.8 | +++ | +++ | 1.8 |
| D | 1.4 | 1.1 | 2.1 | 3.0 | +++ | +++ | +++ |
| E | 3.4 | 2.8 | | | | | 4.3 |
| F | 1.7 | 1.3 | | | | | 4.5 |
| global median | 2.8 | 2.0 | 2.6 | 2.9 | +++ | +++ | 3.3 |

(a) Ratios between median performance values.

| | Local search | | Policy search | | Self-modeling | | reference gait |
|---|---|---|---|---|---|---|---|
| | tests | time | tests | time | time | tests | |
| A | +59 | +45 | +53 | +59 | +64 | +72 | - 2 |
| B | +34 | +33 | +34 | +34 | +63 | +42 | +35 |
| C | +29 | +18 | +29 | +29 | +63 | +74 | +20 |
| D | + 9 | + 2 | +16 | +20 | +38 | +30 | +30 |
| E | +46 | +42 | | | | | +50 |
| F | +18 | +10 | | | | | +35 |
| global median | +32 | +26 | +32 | +32 | +63 | +57 | +33 |

(b) Differences between median performance values (cm).

Table 3: Performance improvements of the T-Resilience compared to other algorithms. For ratios, the symbol +++ indicates that the compared algorithm led to a negative or null median value.

Bongard's algorithm mostly fails because of the reality gap between the self-model and the real robot. Optimizing the behavior only in simulation leads – as expected – to controllers that perform well with the self-model but that do not work on the real robot. This performance loss is sometimes high because the controllers make the robot fall of over or go backward.

**Resilience performance (cases B to F).** When the robot is damaged, gaits found with the T-Resilience algorithm are always faster than the reference gait (p-value $= 0.0625$, one-sample Wilcoxon signed rank test).

After the same number of tests (variant $tests$ of each algorithm), gaits obtained with T-Resilience are at least 1.4 times faster than those obtained with the other algorithms (median of 3.0 times) with median performance values from 30 to 65 cm in 3 seconds. These improvements are all stastically significant (p-values $\leq 0.016$) except for the local search in the case D (loss of a hind leg; p-value $= 0.1508$).

After the same running time (variant $time$ of each algorithm), gaits obtained with T-Resilience are also significantly faster (at least 1.3 times; median of 2.8 times; p-values $\leq 0.016$) than those obtained with the other algorithms in cases B, E and F. In cases C (shortened leg) and D (loss of a hind leg), T-Resilience is not statistically different from local search (shortened leg: p-value $= 0.1508$; loss of a hind leg: p-value $= 0.5476$). Nevertheless, these high p-values may stem from the low number of replications (only 5 replications for each algorithm). Moreover, as section 5.4 will show, the execution time of the T-Resilience can be compressed because a large part of the running time is spent in computer simulations. Consequently, depending on the hardware, better performances could be achieved in smaller amounts of time.

For all the tested cases, Bongard's self-modeling algorithm doesn't find any working controllers. We observed that it suffers from two difficulties: the optimized models do not always capture the actual morphology of the robot, and reality gaps between the self-model and the reality (see the comments about the undamaged robot). In the first case, more time and more actions could improve the result. In the second time, a better simulation model could make things better but it is unlikely to fully remove the effect of the reality gap.

**Loss of a leg (case D and E).** When the hind leg is lost (case D), the T-Resilience yields controllers that perform much better than the reference controller. Nevertheless, the performances of the controllers obtained with the T-Resilience are not statistically different from those obtained with the local search. This unexpected result stems from the fact that many of the transfers made the robot tilt down (fast six-legged behaviors optimized on the self-model of the undamaged robot are often unstable without one of the hind legs): in this case, the SLAM algorithm is unreliable (the algorithm often crashed) and we have to discard the distance measurements. In effect, only a dozen of transfers are usable in case D, making the estimation of the transferability function especially difficult. Using more transfers could accentuate the difference between T-Resilience and local search.

If the robot loses a less critical leg (middle leg in case E), it is more stable and the algorithm can conduct informative tests on the robot. The T-Resilience is then able to find fast gaits (about 3 times faster than with the local search).

**Straightness of the trajectories.** For all the damages, the gaits found by T-Resilience do not result in trajectories that are perfectly aligned with the x-axis (Fig. 9, deviations from 10 to 20 cm). The deviations mainly stem from the choice of the performance function and they should not be overinterpreted as a weakness of the T-Resilience algorithm. Indeed, the performance function only rewards the covered distance during 3 seconds and nothing is explicitly rewarding the straightness of the trajectory. At first, it seems intuitive that the fastest trajectories will necessarily be straight. However, the fastest gaits achievable with this specific robot are un-

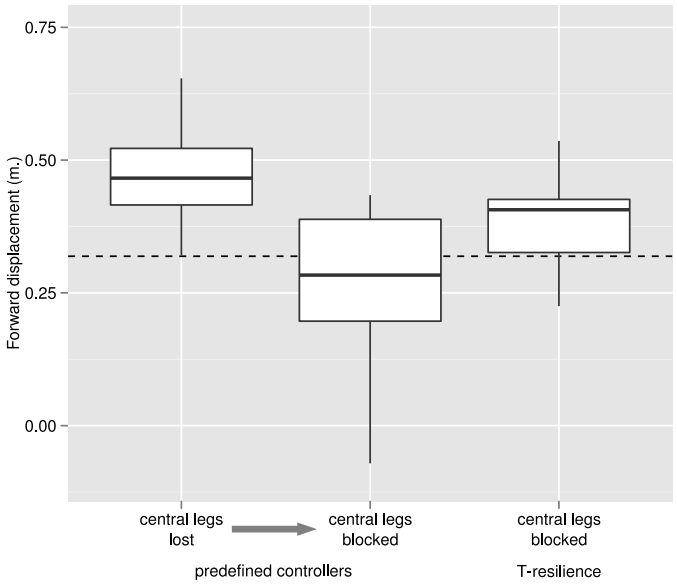



Figure 11: Comparison between the use of predefined controllers and the T-Resilience algorithm (covered distance in meters; 10 replications). (left box) performances of the controllers learned on the robot without its two central legs; (central box) performances of the same controllers on the robot, while the two central legs are blocked; (right box) results obtained with the T-Resilience algorithm on the robot with the two central legs blocked. The dashed line indicates the performances of the reference controller on the robot with the two central legs blocked. Performances are measured with the external CODA scanner.

known and faster gaits may be achievable if the robot is not pointing in the arbitrarily-chosen x-direction. For instance, the trajectory found with the undamaged robot (Fig. 9(a)) deviates from the x-axis, but it has the same final performance as the reference gait, which is perfectly straight. These two gaits cannot be distinguished by the performance function and we have no way to know if both faster and straighter trajectories are possible in our system. The intuition that the straightest trajectories should be the fastest is even more challenged for the damaged robots because the robots are not symmetric anymore. In future work, we will investigate alternative performance functions that encourage straight trajectories.

Moreover, trajectories are actually mostly straight (Fig. 9 and the videos in appendix)) but they do not exactly point to the x-direction. This direction seems to be mainly determined by the position of the robot at the beginning of the experiment: at $t = 0$, each degree of freedom is positioned according to the value of the control function, that is, the robot often starts in the middle of the gait pattern; because this position is non-symmetric and sometimes unstable, we often observed that the first step often makes the robot point in a different direction. Once the gait is started, deviations along one directions are compensated by symmetrical deviations at the next step and the gaits are mostly straight.

## 5.3. Comparison with predefined controllers

The transferability approach found efficient controllers for the robot without the two central legs (right box on Figure 11; median at $0.47$ m). We then tested these controllers on the robot with the two central legs blocked and we observed a significant performance drop (central box on Fig. 11; $0.28$ m vs $0.47$ m; p-value $= 5.4 \times 10^{-4}$). Importantly, among the ten tested controllers, one made the robot goes backward and another one made it fall: depending on the chosen predefined controller, this damage may fully prevent the robot to move.

Facing the same damage, the T-Resilience algorithm found significantly higher-performing controllers than the predefined ones (left box on Fig. 11; $0.41$ m versus $0.28$ m; p-value $= 0.063$). For this damage, anticipating classic failures was therefore significantly less efficient than using the T-resilience. Adding controllers for blocked legs, would only postpone the problem because it is probable that there exists some other damages for which this extended set would not be sufficient. Such an addition would also slow down the adaptation process because each of them need to be tested on the robot.

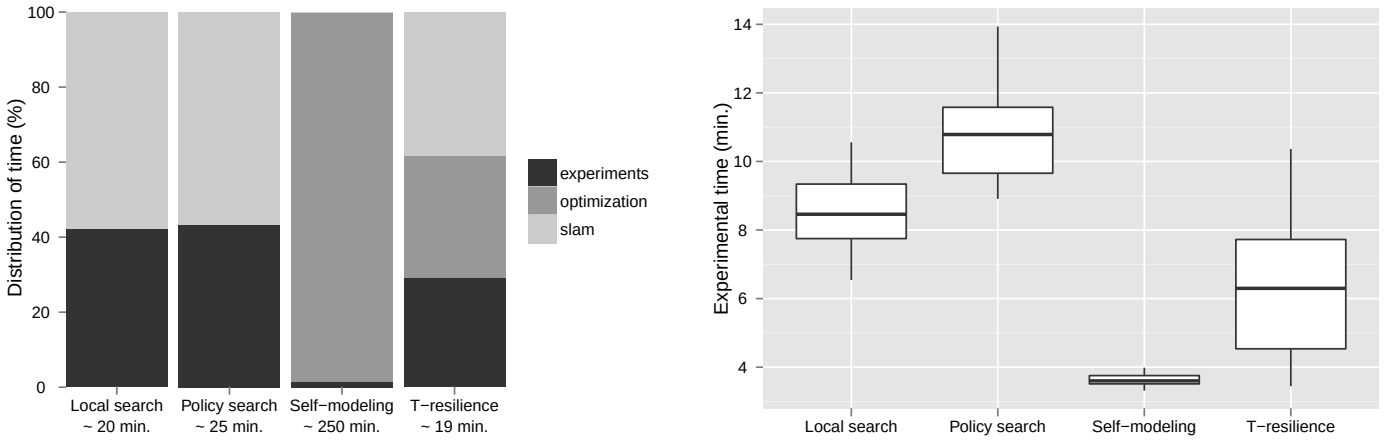
## 5.4. Comparison of durations and experimental time

The running time of each algorithm is divided into experimental time (actual experiments on the robot), sensor processing time (computing the robot's trajectory using RGB-D slam) and optimization time (generating new potential solutions to test on the robot). The median proportion of time allocated to each of this part of the algorithms is pictured for each algorithm on figure 12[8].

The durations of the SLAM algorithm and of the optimization processes both only depend on the hardware specifications and can therefore be substantially reduced by using faster computers or by parallelizing computation. Only experimental time can not easily be reduced. The median proportion of experimental time is 29% for the T-Resilience, whereas both the policy search and the local search leads to median proportions higher than 40% for a similar median duration by run (about 20 minutes). The proportion of experimental time for the self-modeling process is much lower (median value equals to 1%) because it requires much more time for each run (about 250 minutes for each run, in our experiments).

The median experimental time of T-Resilience (6.3 minutes) is significantly lower than those of local search and of policy search (resp. 8.5 and 10.8 minutes, p-values $< 2.5 \times 10^{-4}$). With the expected increases of computational power, this difference will increase each year. The self-modeling process requires significantly lower experimental time (median at 3.6 minutes, p-values $< 1.5 \times 10^{-11}$) because it only test actions that involve a single leg, which is faster than testing a full gait (3 second).

[8]Only test cases A, B, C and D are considered to compute these proportions (5 runs for each algorithm) because the policy search and the self-modeling process are not tested in test cases E and F




(a) Distribution of duration (median duration indicated below the graph).

(b) Experimental time (experiments with the robot).

Figure 12: Distribution of duration and experimental time for each algorithm (median values on 5 runs of test cases A, B, C, D). All the differences between experimental times are statistically significant (p-values $< 2.5 \times 10^{-4}$ with Wilcoxon rank-sum tests).

# 6. Conclusion and discussion

All our experiments show that T-Resilience is a fast and efficient learning approach to discover new behaviors after mechanical and electrical damages (less than 20 minutes with only 6 minutes of irreducible experimental time). Most of the time, T-Resilience leads to gaits that are several times better than those obtained with direct policy search, local search and Bongard's algorithm; T-Resilience never obtained worse results. Overall, T-Resilience appears to be a versatile algorithm for damage recovery, as demonstrated by the successful experiments with many different types of damages. These results validate the combination of the principles that underly our algorithm: (1) using a self-model to transform experimental time with the robot into computational time inside a simulation, (2) learning a transferability function that predicts performance differences between reality and the self-model (instead of learning a new self-model) and, (3) optimizing both the transferability and performance to learn behaviors in simulation that will work well on the real robot, even if the robot is damaged. These principles can be implemented with alternative learning algorithms and alternative regression models. Future work will identify whether better performance can be achieved by applying the same principles with other machine learning techniques.

During our experiments, we observed that the T-Resilience algorithm was less sensitive to the quality of the SLAM than the other investigated learning algorithms (policy gradient and local search). Our preliminary analysis shows that the sensitivity of these classic learning algorithms mostly stems from the fact they optimize the SLAM measurements and not the real performance. For instance, in several of our experiments, the local search algorithm found gaits that make the SLAM algorithm greatly over-estimate the forward displacement of the robot. The T-Resilience algorithm relies only on internal sensors as well. However, these measures are not used to estimate the performance but to compute the transferability values. Gaits that lead to over-estimations of the covered distance have low transferability scores because the measurement greatly differs from the value predicted by the self-model. As a consequence, they are avoided like all the behaviors for which the prediction of the self-model does not match the measurement. In other words, the self-model acts as a "credibility check" of the SLAM measurements, which makes T-Resilience especially robust to sensor inaccuracies. If sensors were redundant, this credibility check could also be used by the robot to continue its mission when a sensor is unavailable.

An obvious limitation of our current results is that it we only investigated straight gaits, whereas mobile robots need to travel around obstacles and change direction. To make the robot walk in any direction, a promising approach is to learn a collection of simple controllers, one for each direction. TBR-learning, a recently introduced algorithm, allows the discovery of such a repertoire of controllers with a single run of the algorithm (Cully and Mouret, 2013a,b). The learning is essentially as fast as learning a single controller for straight walking (Cully and Mouret, 2013a) and it has already been successfully combined with the transferability approach (Cully and Mouret, 2013b). Future work should investigate how TBR-learning performs with a damaged robot.

The T-Resilience algorithm is designed to perform well with an inaccurate self-model, but the better the self-model is, the more efficient the algorithm is. For instance, in the case of the lost hind leg, the stability properties of the robot were too different from those of the self-model and efficient behaviors could hardly be found in a limited amount of time. If the situation of the robot implies that such a strong failure can not be repaired, then the T-Resilience would benefit from an update of the self-model. In some extreme cases, the T-Resilience algorithm could also be unable to find any satisfying controller and it may be mandatory to identify the damage to pursue the mission.

More generally, an algorithm to identify the damage is a natural complement of the T-resilience algorithm. On the short term, the T-resilience quickly provides a good solution after a damage. When the robot has more time or when it can benefit from external diagnoses, it can take the time to understand the damage. For instance, we can imagine a robot in a rescue mission. If it is damaged, it must first go back to its base – or at least to a safe place – and the T-resilience algorithm is designed to allow the robot to do it.

Once it is back to its base, it can use the equipment of the base to diagnose the damages and find a behavior that explicitly takes the damage into account. To identify a new self-model, the robot could also launch Bongard's algorithm and then use the updated model with the T-Resilience algorithm – to avoid reality gaps between the model and the real robot. The combination of T-Resilience and update of the self-model will be addressed in future work.

Overall, learning to predict what behaviors should be avoided is a very general concept that can be applied to many situations in which a robot has to autonomously adapt its behavior. On a higher level, this concept could also share some similarities with what human do when they are injured: if a movement is painful, humans do not fully understand what cause the pain, but they identify the behaviors that cause the pain[9]; once they know that some move are painful, they learn to instinctively avoid them. Humans seem reluctant to permanently change their self-model to reflect what behaviors are possible: people with an immobilized leg still know how to activate their muscles, amputated people frequently report pain in their missing members (Ramachandran and Hirstein, 1998) and dream about themselves in their intact body (Mulder et al., 2008). Humans may therefore learn by combining their self-model with a second model that predicts which behaviors should be avoided, even if they are possible. This model would be similar in essence to a transferability function.

## Acknowledgements

The authors thank Jeff Clune and Stéphane Doncieux for helpful comments and suggestions.

## Funding

This work was supported by the Agence Nationale pour la Recherche [grant number ANR-12-JS03-0009]; a "Université Pierre et Marie Curie - Direction Générale de l'Armement" scholarship to A. Cully; and Polytech'Paris-UPMC.

## References

B. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

T. Barfoot, E. Earon, and G. D'Eleuterio. Experiments in learning distributed control for a hexapod robot. *Robotics and Autonomous Systems*, 54(10):864–872, 2006.

J. G. Bellingham and K. Rajan. Robotics in remote and hostile environments. *Science*, 318(5853):1098–102, November 2007. ISSN 1095-9203.

D. Berenson, N. Estevez, and H. Lipson. Hardware evolution of analog circuits for in-situ robotic fault-recovery. In *Proceedings of NASA/DoD Conference on Evolvable Hardware*, pages 12–19, 2005.

[9] In the same way as learning a transferability function is easier than learning a new self-model, finding the cause of a pain requires a medical doctor whereas a patient can usually predict whether a move will be painful.

J. Bongard. Action-selection and crossover strategies for self-modeling machines. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*, pages 198–205. ACM, 2007.

J. Bongard and H. Lipson. Nonlinear system identification using coevolution of models and tests. *IEEE Transactions on Evolutionary Computation*, 9(4):361–384, 2005.

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

F. Caccavale and L. Villani, editors. *Fault Diagnosis and Fault Tolerance for Mechatronic Systems: Recent Advances*. springer, 2002.

E. Cantu-Paz. *Efficient and accurate parallel genetic algorithms*. Kluwer Academic Publishers Norwell, MA, USA, 2000.

C. Chang and C. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

S. Chernova and M. Veloso. An evolutionary approach to gait learning for four-legged robots. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2562–2567. IEEE, 2004.

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

J. Connell and S. Mahadevan. *Robot learning*. Springer, 1993.

F. Corbato. On Building Systems That Will Fail. *ACM Turing award lectures*, 34(9):72–81, 2007.

A. Cully and J.-B. Mouret. Behavioral repertoire learning in robotics. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*, 2013a.

A. Cully and J.-B. Mouret. Learning to walk in every direction. In *submitted*, 2013b.

K. De Jong. *Evolutionary computation: a unified approach*. MIT Press, 2006.

K. Deb. *Multi-objective optimization using evolutionary algorithms*. John Wiley and Sons, 2001.

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.

F. Delcomyn. The Locomotion of the Cockroach Pariplaneta americana. *Journal of Experimental Biology*, 54(2):443–452, 1971.

X. Ding, Z. Wang, A. Rovetta, and J. Zhu. Locomotion analysis of hexapod robot. *Proceedings of Conference on Climbing and Walking Robots (CLAWAR)*, pages 291–310, 2010.

S. Doncieux, J.-B. Mouret, N. Bredeche, and V. Padois. Evolutionary robotics: Exploring new horizons. In *New Horizons in Evolutionary Robotics: Extended Contributions from the 2009 EvoDeRob Workshop.*, pages 3–25. Springer, 2011.

F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. An evaluation of the RGB-D SLAM system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

K. Goldberg and B. Chen. Collaborative control of robot motion: robustness to error. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 655–660, 2001. ISBN 0-7803-6612-3.

M. Görner and G. Hirzinger. Analysis and evaluation of the stability of a biologically inspired, leg loss tolerant gait for six-and eight-legged walking robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4728–4735, 2010.

J. J. Grefenstette, A. C. Schultz, and D. E. Moriarty. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, pages 241–276, 1999.

C. Hartland and N. Bredeche. Evolutionary robotics, anticipation and the reality gap. In *Robotics and Biomimetics, 2006. ROBIO'06. IEEE International Conference on*, pages 1640–1645. IEEE, 2006.

V. Heidrich-Meisner and C. Igel. Neuroevolution strategies for episodic reinforcement learning. *Journal of Algorithms*, 64(4):152–168, October 2009.

T. Hemker, M. Stelzer, O. Von Stryk, and H. Sakamoto. Efficient walking speed optimization of a humanoid robot. *The International Journal of Robotics Research*, 28(2):303–314, 2009.

M. Hoffmann, H. Marques, A. Arieta, H. Sumioka, M. Lungarella, and R. Pfeifer. Body Schema in Robotics: A Review. *IEEE Transactions on Autonomous Mental Development*, 2(4):304–324, 2010.

O. Holland and R. Goodman. Robots with internal models a route to machine consciousness? *Journal of Consciousness Studies*, 10(4-5):4–5, 2003.

H. H. Hoos and T. Stützle. *Stochastic local search: Foundations and applications*. Morgan Kaufmann, 2005.

G. S. Hornby, J. D. Lohn, and D. S. Linden. Computer-automated evolution of an X-band antenna for NASA's Space Technology 5 mission. *Evolutionary computation*, 19 (1):1–23, January 2011. ISSN 1530-9304.

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.

B. Jakimovski and E. Maehle. In situ self-reconfiguration of hexapod robot oscar using biologically inspired approaches. *Climbing and Walking Robots. InTech*, 2010.

N. Jakobi, P. Husbands, and I. Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. *Proceedings of the European Conference on Artificial Life (ECAL)*, pages 704–720, 1995.

S. Kajita and B. Espiau. *Handbook of Robotics*, chapter Legged Robots, pages 361–389. Springer, 2008.

D. Katić and M. Vukobratović. Survey of intelligent control techniques for humanoid robots. *Journal of Intelligent & Robotic Systems*, 37(2):117–141, 2003.

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.

G. Klaus, K. Glette, and J. Tørresen. A comparison of sampling strategies for parameter estimation of a robot simulator. *Simulation, Modeling, and Programming for Autonomous Robots*, pages 173–184, 2012.

J. Kober and J. Peters. Reinforcement learning in robotics: A survey. In *Reinforcement Learning: State of the Art*, pages 579–610. Springer, 2012.

J. Kober and J. Peters. Imitation and Reinforcement Learning – Practical Learning Algorithms for Motor Primitives in Robotics. *IEEE Robotics and Automation Magazine*, 17(2):1–8, 2010.

N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pages 2619–2624. IEEE, 2004.

S. Koos and J.-B. Mouret. Online discovery of locomotion modes for wheel-legged hybrid robots: a transferability-based approach. In *Proceedings of CLAWAR*, pages 70–77. World Scientific Publishing Co., 2011.

S. Koos, J.-B. Mouret, and S. Doncieux. The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*, 1: 1–25, 2012.

I. Koren and C. M. Krishna. *Fault-tolerant systems*. Morgan Kaufmann, 2007.

C.-M. Lin and C.-H. Chen. Robust fault-tolerant control for a biped robot using a recurrent cerebellar model articulation controller. *Systems, Man, and Cybernetics, Part B: Cybernetics*, 37(1):110–123, 2007.

S. Mahdavi and P. Bentley. An evolutionary approach to damage recovery of robot motion with muscles. *Advances in Artificial Life*, pages 248–255, 2003.

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

T. Metzinger. *Being no one: The self-model theory of subjectivity*. MIT Press, 2004.

T. Metzinger. Self models. *Scholarpedia*, 2(10):4174, 2007.

G. E. Moore. Progress in digital integrated electronics. In *International Electron Devices Meeting*, volume 21, pages 11–13. IEEE, 1975.

K. Mostafa, C. Tsai, and I. Her. Alternative gaits for multiped robots with leg failures to retain maneuverability. *International Journal of Advanced Robotic Systems*, 7(4):31, 2010.

J.-B. Mouret and S. Doncieux. Sferes_v2: Evolvin' in the Multi-Core World. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC)*, pages 4079–4086, 2010.

J.-B. Mouret and S. Doncieux. Encouraging behavioral diversity in evolutionary robotics: an empirical study. *Evolutionary Computation*, 20(1):91–133, January 2012.

J.-B. Mouret, S. Koos, and S. Doncieux. Crossing the reality gap: a short introduction to the transferability approach. In *Proceedings of ALIFE's workshop "Evolution in Physical Systems"*, pages 1–7, 2012.

T. Mulder, J. Hochstenbach, P. Dijkstra, and J. Geertzen. Born to adapt, but not in your dreams. *Consciousness and cognition*, 17(4):1266–71, 2008.

Y. Nakamura, T. Mori, M. Sato, and S. Ishii. Reinforcement learning for a biped robot based on a cpg-actor-critic method. *Neural Networks*, 20(6):723–735, 2007.

A. Nelson, G. Barlow, and L. Doitsidis. Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4):345–370, 2009.

D. Nguyen-Tuong and J. Peters. Model Learning for Robot Control : A Survey. *Cognitive Processing*, 12(4):319–340, 2011.

M. Palmer, D. Miller, and T. Blackwell. An Evolved Neural Controller for Bipedal Walking: Transitioning from Simulator to Hardware. In *Proceedings of IROS' workshop on Exploring new horizons in Evolutionary Design of Robots*, 2009.

G. Parker. Punctuated anytime learning to evolve robot control for area coverage. *Design and Control of Intelligent Robotic Systems*, pages 255–277, 2009.

J. Peters. Policy gradient methods. *Scholarpedia*, 5(10):3698, 2010.

J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697, 2008.

E. Prassler and K. Kosuge. *Handbook of Robotics*, chapter Domestic Robotics, pages 1253–1281. Springer, 2008.

C. Pretorius, M. du Plessis, and C. Cilliers. Simulating robots without conventional physics: A neural network approach. *Journal of Intelligent & Robotic Systems*, pages 1–30, 2012.

Z. Qu, C. M. Ihlefeld, Y. Jin, and A. Saengdeejing. Robust fault-tolerant self-recovering control of nonlinear uncertain systems. *Automatica*, 39(10):1763–1771, 2003.

M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source robot operating system. In *Proceedings of ICRA's workshop on Open Source Software*, 2009.

V. Ramachandran and W. Hirstein. The perception of phantom limbs. *Brain*, 121(9):1603–1630, 1998.

U. Saranli, M. Buehler, and D. Koditschek. Rhex: A simple and highly mobile hexapod robot. *The International Journal of Robotics Research*, 20(7):616–631, 2001.

G. Schleyer and A. Russell. Adaptable gait generation for autotomised legged robots. In *Proceedings of Australasian Conference on Robotics and Automation (ACRA)*, 2010.

J. Schmitz, J. Dean, T. Kindermann, M. Schumm, and H. Cruse. A biologically inspired controller for hexapod walking: simple solutions by exploiting physical properties. *The biological bulletin*, 200(2):195–200, 2001.

A. Smola and V. Vapnik. Support vector regression machines. *Advances in neural information processing systems*, 9:155–161, 1997.

A. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004.

A. Sproewitz, R. Moeckel, J. Maye, and A. Ijspeert. Learning to move in modular robots using central pattern generators and online optimization. *The International Journal of Robotics Research*, 27(3-4):423–443, 2008.

S. Steingrube, M. Timme, F. Wörgötter, and P. Manoonpong. Self-organized adaptation of a simple neural circuit enables complex robot behaviour. *Nature Physics*, 6(3):224–230, 2010.

J. Sturm, C. Plagemann, and W. Burgard. Adaptive body scheme models for robust robotic manipulation. In *Robotics: Science and Systems*, 2008.

R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12(22), 2000.

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

A. Toffolo and E. Benini. Genetic diversity as an objective in multi-objective evolutionary algorithms. *Evolutionary Computation*, 11(2):151–167, 2003.

J. Togelius, T. Schaul, D. Wierstra, C. Igel, and J. Schmidhuber. Ontogenetic and phylogenetic reinforcement learning. *Kuenstliche Intelligenz*, pages 30–33, 2009.

A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

M. Visinsky, J. Cavallaro, and I. Walker. Robotic fault detection and fault tolerance: A survey. *Reliability Engineering & System Safety*, 46(2):139–158, January 1994.

K. Vogeley, M. Kurthen, P. Falkai, and W. Maier. Essential functions of the human self model are implemented in the prefrontal cortex. *Consciousness and cognition*, 8(3):343–63, 1999.

J. Weingarten, G. Lopes, M. Buehler, R. Groff, and D. Koditschek. Automated gait adaptation for legged robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pages 2153–2158, 2004.

S. Whiteson. Evolutionary computation for reinforcement learning. In *Reinforcement Learning: State of the Art*, pages 326–355. Springer, 2012.

D. Wilson. Insect walking. *Annual Review of Entomology*, 11 (1):103–122, 1966.

J. Yosinski, J. Clune, D. Hidalgo, S. Nguyen, J. Zagal, and H. Lipson. Evolving Robot Gaits in Hardware: the HyperNEAT Generative Encoding Vs. Parameter Optimization. *Proceedings of the European Conference on Artificial Life (ECAL)*, pages 1–8, 2011.

J. Zagal, J. Ruiz-del Solar, and P. Vallejos. Back to reality: Crossing the reality gap in evolutionary robotics. In *Proceedings of IFAC Symposium on Intelligent Autonomous Vehicles (IAV)*, 2004.

J. Zagal, J. Delpiano, and J. Ruiz-del Solar. Self-modeling in humanoid soccer robots. *Robotics and Autonomous Systems*, 57(8):819–827, 2009.

V. Zykov. *Morphological and behavioral resilience against physical damage for robotic systems*. PhD thesis, Cornell University, 2008.

V. Zykov, J. Bongard, and H. Lipson. Evolving dynamic gaits on a physical robot. In *Proceedings of Genetic and Evolutionary Computation Conference, Late Breaking Paper (GECCO)*, volume 4, 2004.

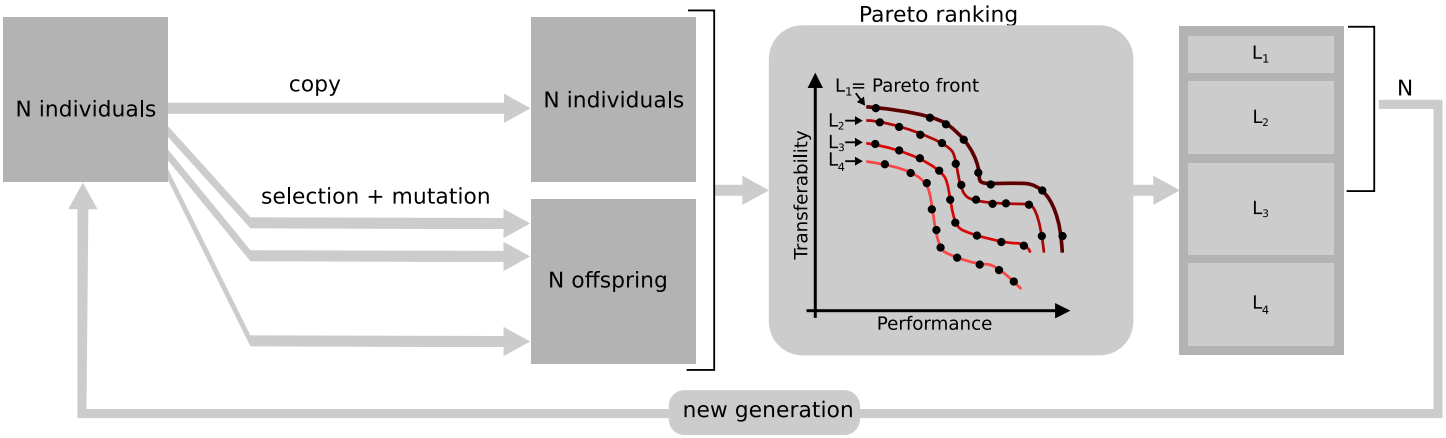



Figure 13: The stochastic multi-objective optimization algorithm NSGA-II (Deb et al., 2002). Starting with a population of $N$ randomly generated individuals, an offspring population of $N$ new candidate solutions is generated using the best candidate solutions of the current population. The union of the offspring and the current population is then ranked according to Pareto dominance (here represented by having solutions in different ranks connected by lines labeled $L_1$, $L_2$, etc.) and the best $N$ candidate solutions form the next generation.

# A. Index to Multimedia Extensions

| Extension | Media type | Description |
|---|---|---|
| 1 | Video | Illustration of the algorithm (case C: front leg shortened) |
| 2 | Video | For each investigated damage: reference controller, median result of T-resilience, and best result of T-resilience |
| 3 | Video | A full T-Resilience run (25 tests, accelerated 2.5 times) in case B (unpowered leg) |
| 4 | Video | A full T-Resilience run (25 tests, accelerated 2.5 times) in case F (both middle and hind legs lost) |
| 5 | Code | Source code (C++) for all the experiments |

# B. NSGA-II

Figure 13 describes the main principle of the NSGA-II algorithm (Deb et al., 2002).

# C. Reference controller

Table 4 shows the parameters used for the reference controller (section 4.1). The amplitude orientation parameters ($\alpha_1^i$) are set to 1 to produce a gait as fast as possible, while the amplitude elevation parameters ($\alpha_2^i$) are set to a small value (0.25) to keep the gait stable. The phase elevation parameters ($\phi_2^i$) define two tripods: 0.25 for legs 0-2-4; 0.75 for legs 1-3-5. To achieve a cyclic motion of the leg, the phase orientation values ($\phi_1^i$) are chosen by subtracting $0.25$ to the phase elevation values ($\phi_2^i$), plus a $0.5$ shift for legs 3-4-5 that are on the left side of the robot.

| Leg number | | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|
| Orientation | $\alpha_1^i$ | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| | $\phi_1^i$ | 0.00 | 0.50 | 0.00 | 0.00 | 0.50 | 0.00 |
| Elevation | $\alpha_2^i$ | 0.25 | 0.25 | 0.25 | 0.25 | 0.25 | 0.25 |
| | $\phi_2^i$ | 0.25 | 0.75 | 0.25 | 0.75 | 0.25 | 0.75 |

Table 4: Parameters of the reference controller.

# D. Implementation details

## D.1. Local search

Our implementation of the local search (algorithm 2) starts from a randomly generated initial controller. A random perturbation $c'$ is derived from the current best controller $c$. The controller $c'$ is next tested on the robot for 3 seconds and the corresponding performance value $\mathcal{F}_{real}(c')$ is estimated with a SLAM algorithm using the RGB-D camera. If $c'$ performs better than $c$, $c$ is replaced by $c'$, else $c$ is kept.

For both the stochastic local search and the policy gradient method (section D.2), a random perturbation $c'$ from a controller $c$ is obtained as follows:

- each parameter $c'_j$ is obtained by adding to $c_j$ a random deviation $\delta_j$, uniformely picked up in $\{-0.25,\ 0,\ 0.25\}$;

- if $c'_j$ is greater (resp. lower) than 1 (resp. 0), it takes the value 1 (resp. 0).

The process is iterated during 20 minutes to match the median duration of the T-Resilience (Table 5; variant $time$). For comparison, the best controller found after 25 real tests is also kept (variant $tests$).

**Algorithm 2** Stochastic local search ($T$ real tests)

$c \leftarrow$ random controller

**for** $i = 1 \rightarrow T$ **do**
  $c' \leftarrow$ random perturbation of $c$
  **if** $\mathcal{F}_{real}(c') > \mathcal{F}_{real}(c)$ **then**
    $c \leftarrow c'$
  **end if**
**end for**

new controller: $c$

## D.2. Policy gradient method

Our implementation of the policy gradient method is based on Kohl and Stone (2004) (algorithm 3). It starts from a randomly generated controller $c$. At each iteration, 15 random perturbations $c'^i$ from this controller are tested for 3 seconds on the robot and their performance values are estimated with the SLAM algorithm, using the RGB-D camera. The number of random perturbations (15) is the same as in (Kohl and Stone, 2004), in which only 12 parameters have to be found. For each control parameter $j$, the average performance $A_{+,j}$ (resp. $A$ or $A_{-,j}$) of the controllers whose parameter value $c'^i_j$ is greater than (resp. equal to or less than) the value of $c_j$ is computed. If $A$ is not greater than both $A_{+,j}$ and $A_{-,j}$, the control parameter $c_j$ is modified as follows:

- $c_j$ is increased by 0.25, if $A_{+,j} > A_{-,j}$ and $c_j < 1$;
- $c_j$ is decreased by 0.25, if $A_{-,j} > A_{+,j}$ and $c_j > 0$.

Once all the control parameters have been updated, the newly generated controller $c$ is used to start a new iteration of the algorithm.

The whole process is iterated 4 times (i.e. 60 real tests; variant $tests$) with a median duration of 24 minutes to match the median duration of the T-Resilience (Table 5). For comparison, the best controller found after 2 iterations (i.e. 30 real tests; variant $time$) is also kept.

**Algorithm 3** Policy gradient method ($T \times S$ real tests)

$c \leftarrow$ random controller

**for** $i = 1 \rightarrow T$ **do**
  $\{c'^1, c'^2, \ldots, c'^S\} \leftarrow S$ random perturbations of $c$
  **for** $j = 1 \rightarrow S$ **do**
    $A_{0,j} =$ average of $\mathcal{F}_{real}(c'^i)$ for $c'^i$ such as $c'^i_j = c_j$
    $A_{+,j} =$ average of $\mathcal{F}_{real}(c'^i)$ for $c'^i$ such as $c'^i_j > c_j$
    $A_{-,j} =$ average of $\mathcal{F}_{real}(c'^i)$ for $c'^i$ such as $c'^i_j < c_j$
    **if** $A_{0,j} > \max(A_{+,j}, A_{-,j})$ **then**
      $c_j$ remains unchanged
    **else**
      **if** $A_{+,j} > A_{-,j}$ **then**
        $c_j = \min(c_j + 0.25,\ 1)$
      **else**
        $c_j = \max(c_j - 0.25,\ 0)$
      **end if**
    **end if**
  **end for**
**end for**

new controller: $c$

## D.3. Self-modeling process (Bongard's algorithm)

Our implementation of the self-modeling process is based on Bongard et al. (2006) (algorithm 4). Unlike the implementation of Bongard et al. (2006), we use internal measurements to assess the consequences of actions. This measure is performed with a 3-axis accelerometer (ADXL345) placed at the center of the robot, thus allowing the robot to measure its orientation.

**Algorithm 4** Self-modeling approach ($T$ real tests)

$pop_{model} \leftarrow \{m^1, m^2, \ldots, m^{S_{model}}\}$ (randomly generated or not)
empty training set of actions $\Omega$

**for** $i = 1 \rightarrow T$ **do**
  selection of the action which maximises variance of predictions in $pop_{model}$
  execution of the action on the robot
  recording of robot's orientation based on internal measurements (accelerometer) | (1) Self-modeling
  addition of the action to the training set $\Omega$
  $N_{model}$ iterations of MOEA on $pop_{model}$ evaluated on $\Omega$
**end for**

selection of the new self-model
$pop_{ctrl} \leftarrow \{c^1, c^2, \ldots, c^{S_{ctrl}}\}$ (randomly generated)

$N_{ctrl}$ iterations of MOEA on $pop_{ctrl}$ in the self-model | (2) Optimization of controllers

selection of the new controller in the Pareto front

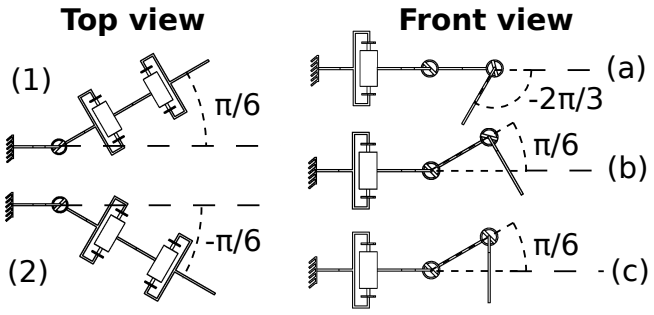



Figure 14: The six possible actions of a leg that can be tested on the robot: (1,a), (2, a), (1,b), (2, b), (1,c), (2, c).

**Robot's model.** The self-model of the robot is a dynamic simulation of the hexapod built with the Open Dynamics Engine (ODE); it is the same model as the one used for T-Resilience experiments. However this self-model is parametrized in order to discover some damages or morphological modifications. For each leg of the robot, the algorithm has to find optimal values for 5 parameters:

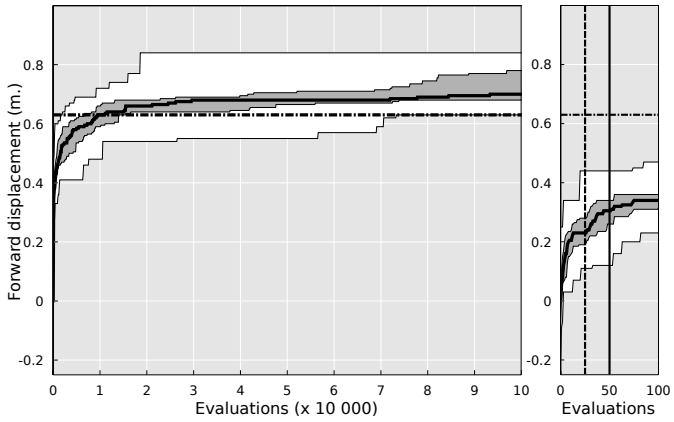
- length of middle part of the leg (float)
- length of the terminal part of the leg (float)
- activation of the first actuator (boolean)
- activation of the second actuator (boolean)
- activation of the last actuator (boolean)

The length parameters have 6 different values: $\{0, 0.5, 0.75, 1, 1.25, 1.5\}$, which represents a scale factor with respect to the original size. If the length parameter of one part is zero, the part is deleted in the simulation and all other parts only attached to it are deleted too. We therefore have a model with 30 parameters.

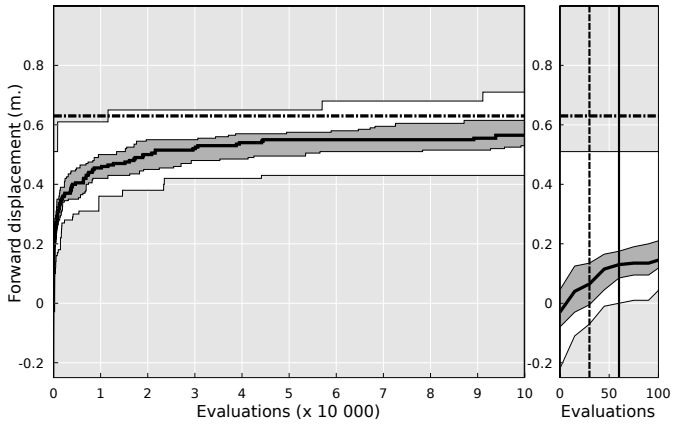
**Action set.** As advised by Bongard (2007) (variant II), we use a set of actions where each action uses only one leg. The first servo has 2 possible positions (1,2): $-\pi/6$ and $\pi/6$. For each of these two positions, we have 3 possible actions (a,b,c) as shown on Figure 14. There are consequently 6 possible actions for each leg, that is, 36 actions in total.

**Parameters.** A population of 36 models is evolved during 2000 generations. The initial population is randomly generated for the initial learning scenario. For other scenarios, the population is initialized with the self-model of the undamaged robot. A new action is tested every 80 generations, which leads to a total of 25 actions tested on the real robot. Applying a new action on the robot implies making an additional simulation for each model at each generation, leading to arithmetic progression of the number of simulation needed per generation. Moreover, $36 \times 36$ additional simulations are needed each time a new action has to be selected and transferred (the whole action set applied to the whole population). In total, about one million simulations have been done per run ($(25 \times 26/2) \times 36 \times 80 + 36 \times 36 \times 25 = 968400$).

The self-modeling process is iterated 25 times (i.e. 25 real tests; variant $tests$) before the optimization of controllers occurs, which leads to a median duration of 250 minutes on overall. (Table 5). For comparison, the best controller optimized with the self-model obtained after 25 minutes of self-modeling is also kept (i.e. after 11 real tests; variant $time$).




(a) Local search.




(b) Policy search.

Figure 15: Performances obtained with the local search (a) and the policy search (b) in simulation (40 runs; $10^5$ evaluations). Thick black curves depict median performance values, dark areas are delimited by first and third quartiles and light areas by lower and upper bounds. Horizontal dashed lines depict the performance of the reference controller in simulation. Figures on the right show the progression of performance values during the first 100 evaluations. Median number of evaluations used in our experiments on the robot for the $tests$ and the $time$ variants are respectively depicted by vertical dashed lines and vertical solid lines. Local search and policy gradient search are both able to find good controllers, provided that they are executed during enough iterations.

# E. Validation of the implementations

To ensure that the observed poor performances are not caused by an implementation error, the local search and the policy search have been tested in simulation with higher numbers of evaluations. Each algorithm has been executed 40 times with $10^5$ evaluations on the simulated hexapod robot, which is used as a self-model with the T-Resilience algorithm. Results are depicted on Figure 15.

These experiments in simulation demonstrate that the small number of evaluations is the cause of the poor performances of these two algorithms in our experiments (cases A to E). Walking controllers are achieved after about $1,000$ evaluations (median performance greater than $0.4$ m). After $2 \times 10^4$ evaluations, both algorithms converge to behaviors with good performances (figure 15; median performances $0.66$ m for local search and $0.50$ m for policy search). These performances have to be balanced with the high number of required evaluations that is most of the time not feasible with a real robot and not compatible with our damage recovery problem. In our experiments with the real robot (section 4), we only performed between 25 and 60 evaluations, which is not enough for the algorithms to find efficient controllers, even in simulation (figure 15; median performances $0.23$ m after 25 evaluations and $0.30$ m after 50 evaluations for local search; $0.06$ m after 30 evaluations and $0.13$ m after 60 evaluations for policy search).

These results indicate that the poor performances observed with both algorithms in our experiments (cases A to E) are mainly caused by low numbers of evaluations performed on the robot.

## F. Median durations and number of tests

| Algorithms | Median duration (min.) | Median number of real tests |
|---|---|---|
| Local search | 20 (10) | 50 (25) |
| Policy search | 25 (13) | 60 (30) |
| T-Resilience | 19 (19) | 25 (25) |
| Self-modeling | 25 (250) | 11 (25) |

Table 5: Median duration and median number of real tests on the robot during a full run for each algorithm, for the "time" variant. Number in parenthesis correspond to the "tests" variant.

## G. Statistical tests

| | Local search | | Policy search | | Self-modeling | | Ref. |
|---|---|---|---|---|---|---|---|
| | tests | time | tests | time | time | tests | |
| A | 0.008 | 0.008 | 0.008 | 0.008 | 0.008 | 0.008 | 1.000 |
| B | 0.008 | 0.016 | 0.016 | 0.016 | 0.008 | 0.008 | 0.063 |
| C | 0.016 | 0.151 | 0.008 | 0.008 | 0.008 | 0.008 | 0.063 |
| D | 0.151 | 0.548 | 0.016 | 0.087 | 0.063 | 0.008 | 0.063 |
| E | 0.008 | 0.008 | | | | | 0.063 |
| F | 0.008 | 0.063 | | | | | 0.063 |

Table 6: Statistical significance when comparing performances between the T-Resilience and the other algorithms (Ref. corresponds to the reference gait). P-values are computed with Wilcoxon rank-sum tests.