Robots that can adapt like natural animals

Article · July 2014

Source: arXiv

3 authors:

Antoine Cully
Imperial College London

Jeff Clune
University of Wyoming

Jean-Baptiste Mouret
National Institute for Research in Computer Science and Control

Robots that can adapt like natural animals

Antoine Cully[1,2], Jeff Clune[3], and Jean-Baptiste Mouret[1,2,*]

While animals can quickly adapt to a wide variety of injuries, current robots cannot "think outside the box" to find a compensatory behavior when damaged: they are either limited to the contingency plans provided by their designers, or need many hours to search for suitable compensatory behaviors. We introduce an intelligent trial and error algorithm that allows robots to adapt to damage in less than 2 minutes, thanks to intuitions that they develop before their mission and experiments that they conduct to validate or invalidate them after damage. The result is a creative process that adapts to a variety of injuries, including damaged, broken, and missing legs. This new technique will enable more robust, effective, autonomous robots and suggests principles that animals may use to adapt to injury.

Robots have transformed the economics of many industries, most notably manufacturing (*2*), and have the power to deliver tremendous benefits to society, such as in search and rescue (*3*), disaster response (*4*), health care (*5*), and transportation (*6*). They are also invaluable tools for scientific exploration, whether of distant planets (*7, 8*), deep oceans (*9*), animal behavior (*10, 11*), or neurobiology (*12*).

As robots leave the controlled environments of factories to autonomously function in unpredictable environments, they will have to respond to the inevitable fact that they will become damaged. Robots presently pale in comparison to natural animals in their ability to invent compensatory behaviors after an injury (Fig. 1A).

Current damage recovery in robots typically involves two phases: self-diagnosis and then selecting the best, pre-designed contingency plan (*13–18*). Such self-diagnosing robots are expensive to manufacture, due to the high cost of self-monitoring sensors, and are difficult to design, because robot engineers cannot foresee every possible situation (*18*): this approach often fails either because the diagnosis is incorrect (*14, 17*) or because an appropriate contingency plan is not provided (*18*).

[1]Sorbonne Universités, UPMC Univ Paris 06, UMR 7222, ISIR, F-75005, Paris, France
[2]CNRS, UMR 7222, ISIR, F-75005, Paris, France
[3]University of Wyoming, Laramie, WY, USA
[*]Corresponding author. E-mail: mouret@isir.upmc.fr

Injured animals respond differently: they learn by trial and error how to compensate for damage (e.g. learning which limp minimizes pain) (*19, 20*). Similarly, trial-and-error learning algorithms could allow robots to creatively discover compensatory behaviors, i.e. without being limited to their designers' assumptions about how damage may occur and how to compensate for each damage type. However, state-of-the-art learning algorithms are impractical because of the "curse of dimensionality" (*21, 22*): the fastest algorithms constrain the search to a few behaviors (e.g. tuning only 2 parameters, requiring 5-10 minutes) or require human demonstrations (*22, 23*). Algorithms without these limitations take several hours (Tables S4, S5). Damage recovery would be much more practical and effective if robots adapted as creatively and quickly as animals (e.g. in minutes), and without expensive self-diagnosing sensors.

The repertoire is created with a simulation of the robot, which either can be a standard physics simulator or can be automatically discovered (*14*). The robot's designers have only to describe the dimensions of the space of possible behaviors and a performance measure. For instance, walking gaits could be described by how much each leg is involved in the gait (a behavioral measure) and speed (a performance measure). To create the repertoire, an optimization algorithm simultaneously searches for a high-performing solution for each point in the behavioral space (Figs. 2A,B and S1). This step requires simulating millions of behaviors, but needs to be performed only once per robot design before deployment (*1*).

A low confidence is assigned to the predicted performance of behaviors stored in this repertoire because they have not been tried in reality (Figs. 2B and S1). During the robot's mission, if it senses a performance drop, it selects the most promising behavior in its repertoire, tests it, and measures its performance. The robot subsequently updates its prediction for that behavior and nearby behaviors, assigns high confidence to these predictions (Figs. 2C and S1), and continues the selection/test/update process until it finds a satisfactory compensatory behavior (Figs. 2D and S1).

All of these ideas are technically captured via a Gaussian process model (*1, 24*), which approximates the performance function using the already acquired data, and a Bayesian optimization procedure (*1, 25, 26*) (Fig. S1), which exploits this model to find the maximum of the performance function. The robot selects which behaviors to test by maximizing an *information acquisition function* that balances exploration (selecting points whose performance is uncertain) and exploitation (selecting points whose performance is expected to be high) (*1*). The selected behavior is tested on the physical robot and the actual performance is recorded. The algorithm updates the expected performance of the tested behavior and lowers the uncertainty about it. These updates are propagated to neighboring solutions in the behavioral space by updating the Gaussian process (*1*). These updated performance and confidence distributions affect which behavior is tested next. This select-test-update loop repeats until the robot finds a behavior whose measured performance is greater than 90% of the best performance predicted for any behavior in the repertoire (*1*).

We test our algorithm on a hexapod robot that needs to walk as fast as possible (Fig. 1B, D). The robot has 18 motors, an onboard computer, and a depth camera that allows the robot to estimate its walking speed (*1*). The gait is parametrized by 36 real-valued parameters that describe the amplitude of oscillation, the phase shift, and the duty cycle for each joint (*1*). The behavior space is 6-dimensional, where

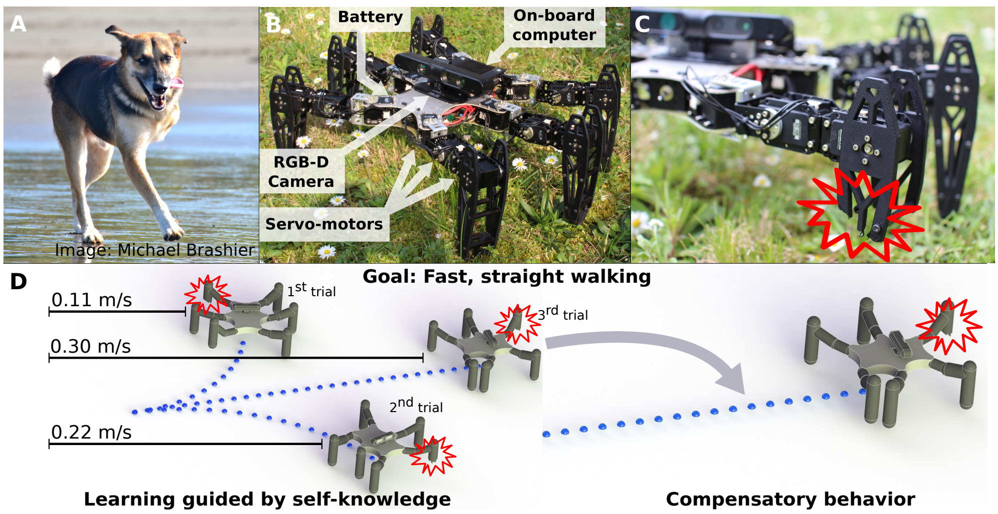

Fig. 1. With Intelligent Trial and Error, robots, like animals, can quickly adapt to recover from damage. (A) Most animals can find a compensatory behavior after an injury. Without relying on *predefined* compensatory behaviors, they learn how to avoid behaviors that are painful or no longer effective. **(B)** An undamaged, hexapod robot. **(C)** One type of damage the experimental hexapod has to cope with (broken leg). **(D)** After damage occurs, in this case making the robot unable to walk straight, damage recovery via Intelligent Trial and Error begins. The robot tests different types of behaviors from an automatically generated behavioral repertoire. After each test, the robot updates its predictions of which behaviors will perform well despite the damage. This way, the robot rapidly discovers an effective compensatory behavior.

each dimension is the proportion of time the i^{th} leg spends in contact with the ground (i.e. the duty factor) (*1, 2*).

The created behavioral repertoire contains approximately 13,000 different gaits (*1*). We tested our robot in six different states: undamaged (Fig. 3A:C1), with four different structural failures (Fig. 3A:C2-C5), and with a temporary leg repair (Fig. 3A:C6). We compare the walking speed of resultant gaits with a widely-used, classic, hand-designed tripod gait (*1, 2*). For each of the 6 damage conditions, we ran our damage recovery phase 5 times for each of 8 independently generated behavioral repertoires, leading to $6 \times 5 \times 8 = 240$ experiments in total.

When the robot is undamaged (Fig. 3A:C1), our approach yields dynamic gaits that are 30% faster than the classic reference gait (Fig. 3B, median 0.32, 5^{th} and 95^{th} percentiles [0.26; 0.36] m/s vs. 0.24), suggesting that Intelligent Trial and Error is a good search algorithm for automatically producing successful robot behaviors, putting aside damage recovery.

In all the damage scenarios, the reference gait is no longer effective (~0.04 m/s for the four damage conditions, Fig. 3B:C2-C5). After Intelligent Trial and Error adaptation, the compensatory gaits are between 3 and 7 times more efficient than the reference gait for that damage condition (in m/s, C2: 0.24 [0.18; 0.31] vs. 0.04; C3: 0.22 [0.18; 0.26] vs. 0.03; C4: 0.21 [0.17; 0.26] vs. 0.04; C5: 0.17 [0.12; 0.24] vs. 0.05; C6: 0.3 [0.21; 0.33]).

These experiments demonstrate that Intelligent Trial and Error allows the robot to both initially learn fast gaits and to reliably recover after physical damage. Additional experiments reveal that these capabilities are substantially faster than state-of-the-art algorithms (Table S4,S5, and supplementary text S1), and that Intelligent Trial and Error can help with another major challenge in robotics: adapting to new environments (supplementary text S2). On the undamaged or repaired robot, Intelligent Trial and Error learns a walking gait in less than 30 seconds (Fig. 3C, undamaged: 24 [16; 41] seconds, 3 [2; 5] physical tests, repaired: 29 [16; 82] seconds, 3.5 [2; 10] tests), which is 8 to 3000 times faster than state-of-the-art techniques (Table S4). For the four damage scenarios, the robot adapts in approximately one minute (66 [24; 134] seconds, 8 [3; 16] tests), which is 15 to 240 times faster than the state-of-the-art (Table S5). Additional experiments show that reducing the high-dimensional space to a low-dimensional behavior space is the key component for intelligent trial and error: standard Bayesian optimization in the original parameter space does not find working controllers (Supplementary text S1).

We investigated how the repertoire is updated when the robot loses a leg (Fig. 3A:C4). Initially the repertoire predicts large areas of high performance. During adaptation, these areas disappear because the behaviors do not work well on the damaged robot (Figs. 4 and S9). Intelligent Trial and Error quickly identifies one of the few remaining, high-performance behaviors (Figs. 4 and S9).

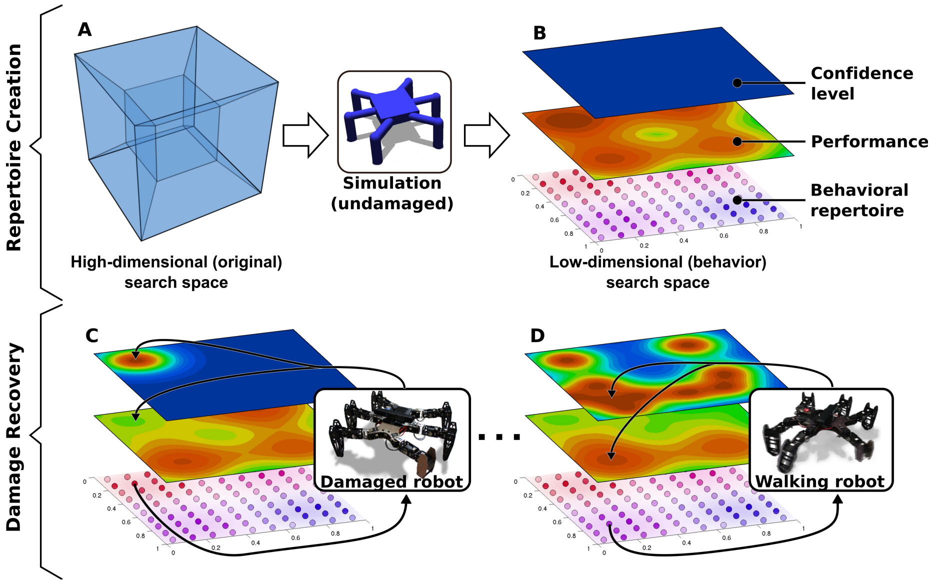

Fig. 2. (A & B). Behavioral repertoire creation. A user reduces a high-dimensional search space to a low-dimensional behavior space by defining dimensions along which behaviors vary. In simulation, the high-dimensional space is searched to produce a repertoire of behaviors (colored spheres) that perform well at each point in this low-dimensional behavioral space. In the example in this paper, the behavior space is six-dimensional: the portion of time that each leg of a hexapod robot is in contact with the ground. The confidence regarding the accuracy of the predicted performance for each behavior in the repertoire is initially low because no tests on the physical robot have been conducted. **(C & D) Damage recovery.** After damage, the robot selects a promising behavior, tests it, updates the predicted performance of that behavior in the repertoire, and sets a high confidence on this performance prediction. The predicted performances of nearby behaviors–and confidence in those predictions–are likely to be similar to the tested behavior and are thus updated via Gaussian processes. This select/test/update loop is repeated until a tested behavior on the physical robot performs better than 90% of the best predicted performance in the repertoire, a value that can decrease with each test (Fig. S1). The algorithm that selects which behavior to test next balances between choosing the behavior with the highest predicted performance and behaviors that are different from those tested so far. Overall, the Intelligent Trial and Error approach presented here rapidly locates which types of behaviors are least affected by the damage to find an effective, compensatory behavior.

While natural animals do not use the specific algorithm we present, there are parallels between Intelligent Trial and Error and animal learning. Like animals, our robot does not have a predefined strategy for how to cope with each of a set of possible damages: in the face of a new injury, it exploits its intuitions about how its body works to experiment with different behaviors to find what works best. Also like animals (*27*), Intelligent Trial and Error allows the quick identification of working behaviors with a few, diverse tests instead of trying behaviors at random or trying small modifications to the best behavior found so far. Additionally, the Bayesian optimization procedure followed by our robot appears similar to the technique employed by humans when they optimize an unknown function (*26, 28*), and there is strong evidence that animal brains learn probability distributions, combine them with prior knowledge, and act as Bayesian optimizers (*29, 30*).

An additional parallel is that Intelligent Trial and Error primes the robot for creativity during a motionless period, after which the generated ideas are tested. This process is reminiscent of the finding that some animals start the day with new ideas that they may quickly disregard after experimenting with them (*31*), and more generally, that sleep improves creativity on cognitive tasks (*32, 33*). A final parallel is that the simulator and Gaussian process components of Intelligent Trial and Error are two forms of predictive models, which are known to exist in animals (*14, 34*). All told, we've shown that combining pieces of nature's algorithm, even if differently assembled, moves robots more towards animals by endowing them with the ability to rapidly adapt to unforeseen circumstances.

References and Notes

1. Materials and methods are available as supplementary material on science online.
2. B. Siciliano, O. Khatib, *Springer handbook of robotics* (Springer, 2008).
3. R. R. Murphy, *Robotics & Automation Magazine, IEEE* **11**, 50 (2004).
4. K. Nagatani, *et al.*, *Journal of Field Robotics* **30**, 44 (2013).
5. E. Broadbent, R. Stafford, B. MacDonald, *International Journal of Social Robotics* **1**, 319 (2009).
6. S. Thrun, *et al.*, *Journal of field Robotics* **23**, 661 (2006).
7. J. G. Bellingham, K. Rajan, *Science* **318**, 1098 (2007).
8. S. Squyres, *Roving Mars: Spirit, Opportunity, and the exploration of the red planet* (Hyperion, 2005).
9. D. R. Yoerger, M. Jakuba, A. M. Bradley, B. Bingham, *The International Journal of Robotics Research* **26**, 41 (2007).
10. J. Halloy, *et al.*, *Science* **318**, 1155 (2007).
11. G. L. Patricelli, J. A. C. Uy, G. Walsh, G. Borgia, *Nature* **415**, 279 (2002).
12. B. Webb, *Nature* **417**, 359 (2002).
13. P. M. Frank, *Automatica* **26**, 459 (1990).
14. J. Bongard, V. Zykov, H. Lipson, *Science* **314**, 1118 (2006).
15. V. Verma, G. Gordon, R. Simmons, S. Thrun, *Robotics & Automation Magazine* **11**, 56 (2004).
16. A. Goye, F. Wiesel, *Information Technology* **47** (2005).

17. W. G. Fenton, T. M. McGinnity, L. P. Maguire, *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* **31**, 269 (2001).
18. J. Kluger, J. Lovell, *Apollo 13* (Mariner Books, 2006).
19. J. Kirpensteijn, R. Van Der Bos, N. Endenburg, *Veterinary record* **144**, 115 (1999).
20. S. L. Jarvis, *et al.*, *American journal of veterinary research* **74**, 1155 (2013).
21. D. Donoho, *AMS Math Challenges Lecture* pp. 1–32 (2000).
22. J. Kober, J. A. Bagnell, J. Peters, *The International Journal of Robotics Research* **32**, 1238 (2013).
23. B. D. Argall, S. Chernova, M. Veloso, B. Browning, *Robotics and autonomous systems* **57**, 469 (2009).
24. C. E. Rasmussen, C. K. I. Williams, *Gaussian processes for machine learning* (MIT Press, 2006).
25. J. Mockus, *Bayesian approach to global optimization: theory and applications* (Kluwer Academic, 2013).
26. A. Borji, L. Itti, *Advances in Neural Information Processing Systems 26 (NIPS)* (2013), pp. 55–63.
27. S. Benson-Amram, K. E. Holekamp, *Proceedings of the Royal Society B: Biological Sciences* **279**, 4087 (2012).
28. T. L. Griffiths, C. Lucas, J. Williams, M. L. Kalish, *Advances in Neural Information Processing Systems 21 (NIPS)* (2009), pp. 553–560.
29. A. Pouget, J. M. Beck, W. J. Ma, P. E. Latham, *Nature neuroscience* **16**, 1170 (2013).
30. K. P. Körding, D. M. Wolpert, *Nature* **427**, 244 (2004).
31. S. Derégnaucourt, P. P. Mitra, O. Fehér, C. Pytte, O. Tchernichovski, *Nature* **433**, 710 (2005).
32. U. Wagner, S. Gais, H. Haider, R. Verleger, J. Born, *Nature* **427**, 352 (2004).
33. D. J. Cai, S. A. Mednick, E. M. Harrison, J. C. Kanady, S. C. Mednick, *Proceedings of the National Academy of Sciences* **106**, 10130 (2009).
34. M. Ito, *Nature Reviews Neuroscience* **9**, 304 (2008).

Supplementary Materials

Material and Methods
Figures S1 to S9
Tables S1 to S6
Text S1 and S2
References (35 to 78)
Movies S1 and S2

Acknowledgements
Thanks to Stéphane Doncieux, Nicolas Brecheche, Shimon Whiteson, Roberto Calandra, Jacques Droulez, Pierre Bessière, Danesh Tarapore, Florian Lesaint, Charles Thurat, Jingyu Li, Joost Huizinga, Roby Velez, and Anh Nguyen for helpful feedback and discussions. Thanks to Michael Brashier for the photo of the three-legged dog. This work has been funded by the ANR Creadapt project, ANR-12-JS03- 0009 and a DGA scholarship to A. Cully.

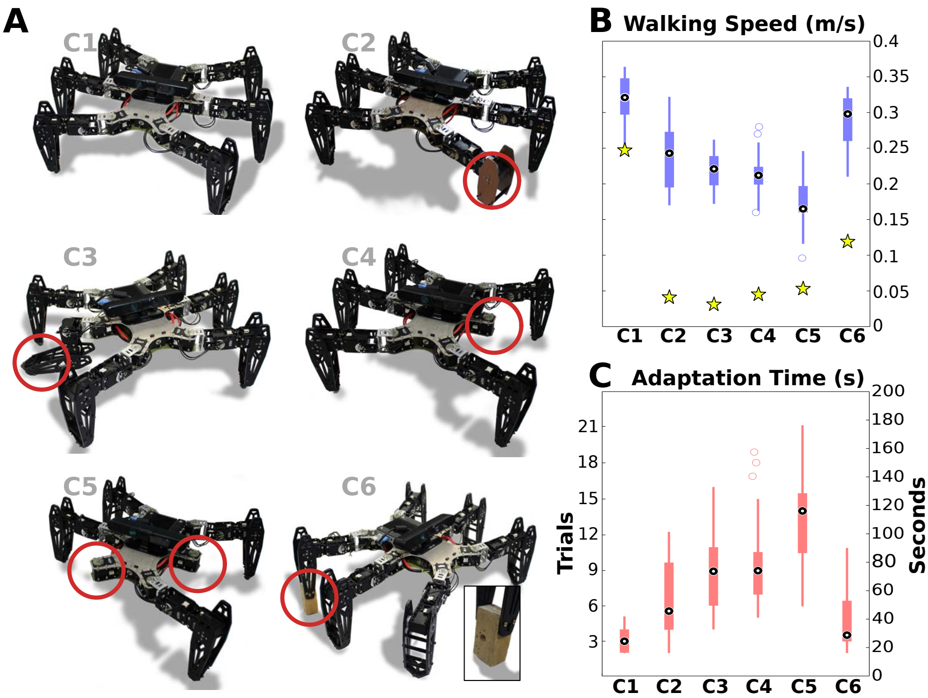

Fig. 3. (A). Conditions tested on the physical robot. (C1) The undamaged robot. **(C2)** One leg is shortened by half. **(C3)** One leg is unpowered. **(C4)** One leg is missing. **(C5)** Two legs are missing. **(C6)** A temporary, makeshift repair to the tip of one leg. **(B) Performance after adaptation.** Box plots represent Intelligent Trial and Error. Yellow stars represent the performance of the handmade reference tripod gait (*1*). **(C) Time required to adapt.** Box plots represent Intelligent Trial and Error. The central mark is the median, the edges of the box are the 25th and 75th percentiles, the whiskers extend to the most extreme data points not considered outliers, and outliers are plotted individually. Each condition is tested with 8 independently created behavioral repertoires and replicated 5 times (i.e. 40 experiments per damage condition).

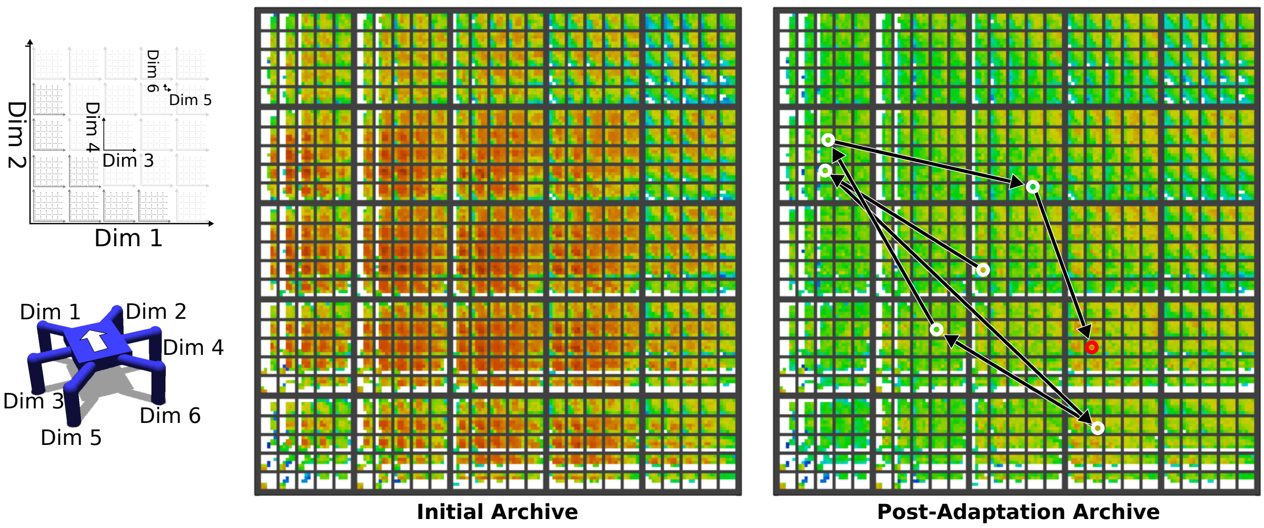

Fig. 4. An example behavioral repertoire. This repertoire stores high-performing behaviors at each point in a six-dimensional behavior space. Each dimension is the portion of time that each leg is in contact with the ground. The behavioral space is discretized at five values for each dimension (0; 0.25; 0.5; 0.75; 1). Each colored pixel represents the highest-performing behavior discovered during repertoire creation at that point in the behavior space. The matrices visualize the behavioral space in two dimensions according to the legend in the top-left. The left matrix is one of the eight pre-adaptation repertoires. During adaptation, the repertoire is updated as tests are conducted (in this case, in the damage condition where the robot is missing one leg: Fig. 3A:C4). The right matrix shows the state of the repertoire after a compensatory behavior was discovered. The white circles represent the order in which behaviors were tested on the physical robot. The red circle is the final, discovered, compensatory behavior. Amongst other areas, high-performing behaviors can be found in the first two columns of the third dimension. These columns represent behaviors that least use the central-left leg, which is the leg that is missing.

Supplementary Materials for

“Robots that can adapt like natural animals”

Antoine Cully[1,2], Jeff Clune[3], and Jean-Baptiste Mouret[1,2]

correspondence to: mouret@isir.upmc.fr

This PDF file includes:

Materials and Methods

Figures S1 to S9

Tables S1 to S6

Text S1 and S2

References (35 to 78)

Captions for movies S1 and S2

Other Supplementary Materials for this manuscript:

Movies S1 and S2

Source code for experiments

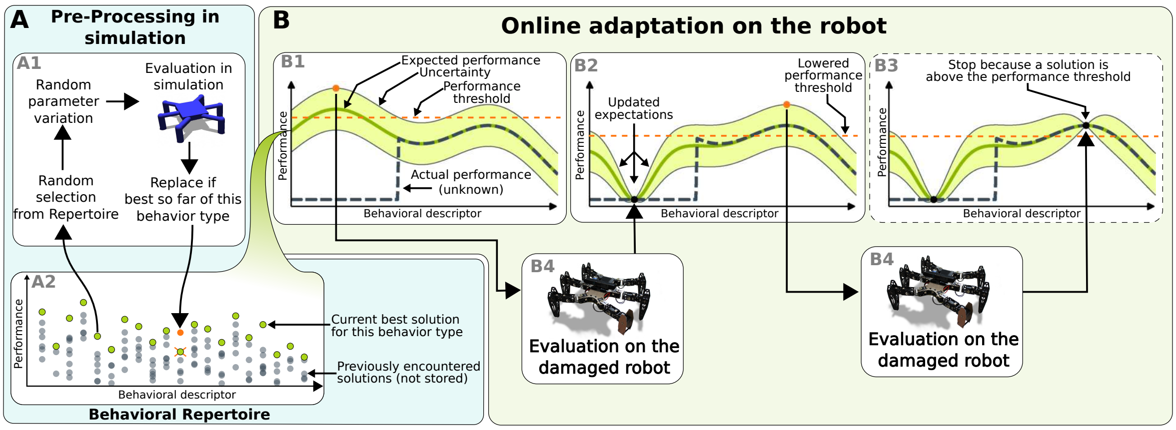

Fig. S1. Overview of the Intelligent Trial and Error Algorithm. (A) Behavioral repertoire creation. After being initialized with random controllers, the behavioral repertoire (A2), which stores the highest-performing controller found so far of each behavior type, is improved by repeating the process depicted in (A1) until newly generated controllers are rarely good enough to be added to the repertoire (here, after 20 million evaluations). This step, which occurs in simulation, is computationally expensive, but only needs to be performed once per robot (or robot design) prior to deployment. In our experiments, creating one repertoire involved 20 million iterations of (A1), which lasted roughly two weeks on one multi-core computer (section S1.7). **(B) Damage recovery.** (B1) Each behavior from the repertoire has an expected performance based on its performance in simulation (dark green line) and an estimate of uncertainty regarding this predicted performance (light green band). The actual performance on the now-damaged robot (black dashed line) is unknown to the algorithm. A behavior is selected to try on the damaged robot. This selection is made by balancing exploitation—trying behaviors expected to perform well—and exploration—trying behaviors whose performance is uncertain (section S1.3). Because all points initially have equal, maximal uncertainty, the first point chosen is that with the highest expected performance. Once this behavior is tested on the physical robot (B4), the performance predicted for that behavior is set to its actual performance, the uncertainty regarding that prediction is lowered, and the predictions for, and uncertainties about, nearby controllers are also updated (according to a Gaussian process model, section S1.3), the result of which can be seen in (B2). The process is then repeated until performance on the damaged robot is 90% or greater of the max expected performance for any behavior (B3). This performance threshold (orange dashed line) lowers as the max expected performance (the highest point on the dark green line) is lowered, which occurs when physical tests on the robot underperform expectations, as occurred in (B2).

S1 Materials and Methods

Notation	Meaning	Type
$\mathbf{x}$	A location in a discrete behavioral space (i.e. a type of behavior)	vector
χ	A location in a discrete behavioral space that has been tested on the physical robot	vector
$\mathcal{P}$	Behavioral repertoire (stores performance)	associative table
$\mathcal{C}$	Behavioral repertoire (stores controllers)	associative table
$\mathcal{P}(\mathbf{x})$	Max performance yet encountered at $\mathbf{x}$	scalar
$\mathcal{C}(\mathbf{x})$	Controller currently stored in $\mathbf{x}$	vector
$\chi_{\mathbf{1:t}}$	All previously tested behavioral descriptors at time t	vector of vectors
$\mathbf{P}_{\mathbf{1:t}}$	Performance in reality of all the candidate solutions tested on the robot up to time t	vector
$\mathcal{P}(\chi_{\mathbf{1:t}})$	Performance for all the candidate solutions tested on the robot up to time t	vector
$f()$	Performance function (unknown by the algorithm)	function
σ^2_{noise}	Observation noise (a user-specified parameter)	scalar
$k(\mathbf{x},\mathbf{x})$	Kernel function (see section S1.3)	function
$\mathbf{K}$	Kernel matrix	matrix
$\mathbf{k}$	Kernel vector $[k(\mathbf{x},\chi_1),[k(\mathbf{x},\chi_2),...,[k(\mathbf{x},\chi_t),]$	vector
$\mu_t(\mathbf{x})$	Predicted performance for $\mathbf{x}$ (i.e. the mean of the Gaussian process)	function
$\sigma^2_t(\mathbf{x})$	Standard deviation for $\mathbf{x}$ in the Gaussian process	function

S1.1 Intelligent Trial and Error Algorithm

The Intelligent Trial and Error Algorithm consists of two major steps (Fig. S1): the behavioral repertoire creation step and the damage recovery step (which more generally could be used for any required adaptation, such as learning an initial gait for an undamaged robot, adapting to new environments, etc.). The behavioral repertoire creation step is accomplished via a new algorithm introduced in this paper called multi-dimensional archive of phenotypic elites (MAP-Elites), which is explained in section S1.2. The damage recovery step is accomplished via a new algorithm introduced in this paper called the repertoire-based Bayesian optimization algorithm (RBOA), which is explained in section S1.3.

In pseudo-code:

```
procedure INTELLIGENT TRIAL AND ERROR ALGORITHM
    Before the mission:
        CREATE BEHAVIORAL REPERTOIRE (VIA THE MAP-ELITES ALGORITHM IN SIMULATION)
    while In mission do
        if Significant performance fall then
            DAMAGE RECOVERY STEP (VIA RBOA ALGORITHM)
```

S1.2 Behavioral repertoire creation (via the MAP-Elites algorithm)

The behavioral repertoire is created by a new algorithm we introduce in this paper called the multi-dimensional archive of phenotypic elites (MAP-Elites) algorithm. MAP-Elites searches for the highest-performing solution for each point in a user-defined space: the user chooses the dimensions of the space that they are interested in seeing variation in. For example, when designing robots, the user may be interested in seeing the highest-performing solution at each point in a two-dimensional space where one axis is the weight of the robot and the other axis is the height of the robot. Alternatively, a user may wish to see weight vs. cost, or see solutions throughout a 3D space of weight vs. cost vs. height. Any dimension that can vary could be chosen by the user. There is no limit on the number of dimensions that can be chosen, although it becomes computationally more expensive to fill the map and store it as the number of dimensions increases. It also becomes more difficult to visualize the results. We refer to this user-defined space as the "behavior space", because usually the dimensions of variation measure behavioral characteristics. Note that the behavioral space can refer to other aspects of the solution (as in this example, where the dimensions of variation are physical properties of a robot such as its height and weight).

If the behavior descriptors and the parameters of the controller are the same (i.e. if there is only one possible solution/genome/parameter set/policy/description for each location in the behavioral space), then creating the repertoire is straightforward: one simply needs to simulate the solution at each location in the behavior space and record the performance. However, if it is not known a priori how to produce a controller/parameter set/description that will end up in a specific location in the behavior space (i.e. if the parameter space is of higher dimension than the behavioral space: e.g., in our example, if there are many different robot designs of a specific weight, height, and cost, or if it is unknown how to make a description that will produce a robot with a specific weight, height, and cost), then MAP-Elites is beneficial. It will efficiently search for the highest-performing solution at each point of the low-dimensional behavioral space. It is more efficient than a random sampling of the search space because high-performing solutions are often similar in many ways, such that permuting a high-performing solution of one type can produce a high-performing solution of a different type. For this reason, searching for high-performing solutions of all types simultaneously is much quicker than separately searching for each type. For example, to generate a lightweight, high-performing robot design, it tends to be more effective and efficient to modify an existing design of a light robot rather than randomly generate new designs from scratch or launch a separate search process for each new type of design.

MAP-Elites begins by generating a set of random candidate solutions. It then evaluates the performance of each solution and records where that solution is located in the behavior space (e.g. if the dimensions of the behavior space are the height and weight, it records the height and weight of each robot in addition to its performance). For each solution, if its performance is better than the current solution in the archive at that location in the behavior space, then it is added to the repertoire, replacing the solution in that location. In other words, it is only kept if it is the best of that type of solution, where "type" is defined as a location in the behavior space. There is thus only one solution kept at each location in the behavior space (keeping more could be beneficial, but for computational reasons we only keep one). If no solution is present in the archive at that location, then the newly generated candidate solution is added.

Once this initialization step is finished, Map-Elites enters a loop that is similar to stochastic, population-based, optimization algorithms, such as evolutionary algorithms (*35*): the solutions that are in the archive/repertoire form a population that is improved by random variation and selection. In each generation, the algorithm picks a solution at random via a uniform distribution, meaning that each solution has an equal chance of being chosen. A copy of the selected solution is then randomly mutated to change it in some way, its performance is evaluated, its location in the behavioral space is determined, and it is kept if it outperforms the current occupant at that point in the behavior space (note that mutated solutions may end up in different behavior space locations than their "parents"). This process is repeated until a stopping criterion is met (e.g. after a fixed amount of time has expired). In our experiments, we stopped each MAP-Elites run after 20 million iterations. Because MAP-Elites is a stochastic search process, each resultant map can be different, both in terms of the number of locations in the behavioral space for which a candidate is found, and in terms of the performance of the candidate in each location.

In pseudo-code:

procedure MAP-ELITES ALGORITHM
    ($\mathscr{P} \leftarrow \emptyset, \mathscr{C} \leftarrow \emptyset$)   ▷ Creation of the repertoire (empty N-dimensional grid.)
    tmp $\leftarrow \{\mathbf{c}^1, \mathbf{c}^2, \ldots, \mathbf{c}^{400}\}$ ▷ Randomly generate 400 controllers
    for $\mathbf{c} \in$ **tmp do**
        $\mathbf{x} \leftarrow$behavioral_descriptor(simu($\mathbf{c}$))   ▷ Simulate the controller and record its behavioral descriptor
        $p \leftarrow$performance(simu($\mathbf{c}$))   ▷ Record its performance
        if $\mathscr{P}(\mathbf{x}) = \emptyset$ or $\mathscr{P}(\mathbf{x}) < p$ **then**
            $\mathscr{P}(\mathbf{x}) \leftarrow p$ ▷ Store its performance in the behavioral repertoire according to its behavioral descriptor
            $\mathscr{C}(\mathbf{x}) \leftarrow \mathbf{c}$ ▷ Associate the controller with its behavioral descriptor.
    for iter = $1 \rightarrow I$ **do**   ▷ Repeat during I iterations (here we choose I = 20 million iterations).
        $\mathbf{c} \leftarrow$ random_selection($\mathscr{C}$)   ▷ Randomly select a controller c in the repertoire
        $\mathbf{c}' \leftarrow$ random_variation($\mathbf{c}$)   ▷ Create a randomly modified copy of c
        $\mathbf{x}' \leftarrow$behavioral_descriptor(simu($\mathbf{c}'$))   ▷ Simulate the controller and record its behavioral descriptor
        $p' \leftarrow$performance(simu($\mathbf{c}'$))   ▷ record its performance
        if $\mathscr{P}(\mathbf{x}') = \emptyset$ or $\mathscr{P}(\mathbf{x}') < p'$ **then**
            $\mathscr{P}(\mathbf{x}') \leftarrow p'$ ▷ Store its performance in the behavioral repertoire according to its behavioral descriptor
            $\mathscr{C}(\mathbf{x}') \leftarrow \mathbf{c}'$ ▷ Associate the controller with its behavioral descriptor.
    return repertoire ($\mathscr{P}$ and $\mathscr{C}$)

Simulator. The simulator is a dynamic physics simulation of the undamaged 6-legged robot on flat ground (Fig. S5). We weighted each segment of the leg and the body of the real robot, and we used the same masses for the simulations. The simulator is based on the Open Dynamics Engine (ODE, `http://www.ode.org`).

Performance function. Performance is the speed of the controller averaged over 5 seconds on a simulated version of the robot (Fig. S5).

Random variation. Each parameter of the controller (section S1.5) has a 5% chance of being changed to any value in the set of possible values, with the new value chosen randomly from a uniform distribution over the possible values.

Behavioral descriptor. The behavioral descriptor is a 6-dimensional vector that corresponds to the proportion of time that each leg is in contact with the ground(also called duty factor). When a controller is simulated, the algorithm records at each time step (every 30 ms) whether each leg is in contact with the ground (1: contact, 0: no contact). The result is 6 Boolean time series (C_i for the i^{th} leg). The behavioral descriptor is then computed with the average of each time series:

$$\mathbf{x} = \begin{bmatrix} \frac{\sum_t C_1(t)}{\text{length}(C_1)} \\ \vdots \\ \frac{\sum_t C_6(t)}{\text{length}(C_6)} \end{bmatrix} \tag{1}$$

During the generation of the behavioral repertoire, the behaviors are stored in the repertoire's cells by discretizing each dimension of the behavioral descriptor space with these five values: {0, 0.25, 0.5, 0.75, 1}. During the adaptation phase, the behavioral descriptors are used with their actual values and are thus not discretized.

Table 1. Main parameters of MAP-Elites.

parameters in controller	parameter values (controller)	size of behavioral space	possible behavioral descriptors	iterations
36	0 to 1, with 0.05 increments	6	{0, 0.25, 0.5, 0.75, 1}	20 millions

S1.3 Damage recovery step (via RBOA: the repertoire-based Bayesian optimization algorithm)

The damage recovery step is accomplished via a Bayesian Optimization Algorithm seeded with a behavioral repertoire. We call this approach a repertoire-based Bayesian optimization algorithm, or RBOA.

Bayesian optimization is a model-based, black-box optimization algorithm that is tailored for very expensive objective functions (or cost functions) (*25, 26, 28, 36–38*). As a black-box optimization algorithm, Bayesian optimization searches for the maximum of an unknown objective function from which samples can be obtained (e.g., by measuring the performance of a robot). Like all model-based optimization algorithms (e.g. surrogate-based algorithms (*39–41*), kriging (*42*), or DACE (*43, 44*)), Bayesian optimization creates a model of the objective function with a regression method, uses this model to select the next point to acquire, then updates the model, etc. It is called *Bayesian* because, in its general formulation (*25*), this algorithm chooses the next point by computing a posterior distribution of the objective function using the likelihood of the data already acquired and a prior on the type of function.

Here we use Gaussian process regression to find a model (*24*), which is a common choice for Bayesian optimization (*28, 36, 37, 45*). Gaussian processes are particularly interesting for regression because they not only model the cost function, but also the uncertainty associated with each prediction. For a cost function f, usually unknown, a Gaussian process defines the probability distribution of the possible values $f(\mathbf{x})$ for each point $\mathbf{x}$. These probability distributions are Gaussian, and are therefore defined by a mean (μ) and a standard deviation (σ). However, μ and σ can be different for each $\mathbf{x}$; we therefore define a probability distribution *over functions*:

$$P(f(\mathbf{x})|\mathbf{x}) = \mathcal{N}(\mu(\mathbf{x}), \sigma^2(\mathbf{x})) \tag{2}$$

where $\mathcal{N}$ denotes the standard normal distribution.

To estimate $\mu(\mathbf{x})$ and $\sigma(\mathbf{x})$, we need to fit the Gaussian process to the data. To do so, we assume that each observation $f(\chi)$ is a sample from a normal distribution. If we have a data set made of several observations, that is, $f(\chi_1), f(\chi_2), ..., f(\chi_t)$, then the vector $[f(\chi_1), f(\chi_2), ..., f(\chi_t)]$ is a sample from a *multivariate* normal distribution, which is defined by a mean vector and a covariance matrix. A Gaussian process is therefore a generalization of a n-variate normal distribution, where n is the number of observations. The covariance matrix is what relates one observation to another: two observations that correspond to nearby values of χ_1 and χ_2 are likely to be correlated (this is a prior assumption based on the fact that functions tend to be smooth, and is injected into the algorithm via a prior on the likelihood of functions), two observations that correspond to distant values of χ_1 and χ_2 should not influence each other (i.e. their distributions are not correlated). Put differently, the covariance matrix represents that distant samples are almost uncorrelated and nearby samples are strongly correlated. This covariance matrix is defined via a *kernel function*, called $k(\chi_1, \chi_2)$, which is usually based on the Euclidean distance between χ_1 and χ_2 (see the "kernel function" sub-section below).

Given a set of observations $\mathbf{P}_{1:t} = f(\chi_{1:t})$ and a sampling noise σ^2_{noise}(which is a user-specified parameter, set to 0.001 in all our experiments), the Gaussian process is computed as follows (*24, 37*):

$$\begin{aligned}
&P(f(\mathbf{x})|\mathbf{P}_{1:t}, \mathbf{x}) = \mathcal{N}(\mu_t(\mathbf{x}), \sigma_t^2(\mathbf{x}))\\
&\text{where:}\\
&\mu_t(\mathbf{x}) = \mathbf{k}^\intercal \mathbf{K}^{-1} \mathbf{P}_{1:t}\\
&\sigma_t^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) + \sigma^2_{noise} - \mathbf{k}^\intercal \mathbf{K}^{-1} \mathbf{k}\\
&\mathbf{K} = \begin{bmatrix} k(\chi_1, \chi_1) & \cdots & k(\chi_1, \chi_t)\\ \vdots & \ddots & \vdots\\ k(\chi_t, \chi_1) & \cdots & k(\chi_t, \chi_t) \end{bmatrix} + \sigma^2_{noise} I\\
&\mathbf{k} = \begin{bmatrix} k(\mathbf{x}, \chi_1) & k(\mathbf{x}, \chi_2) & \cdots & k(\mathbf{x}, \chi_t) \end{bmatrix}
\end{aligned} \tag{3}$$

Our implementation of Bayesian optimization uses this Gaussian process model to search for the maximum of the objective function $f(\mathbf{x})$, $f(\mathbf{x})$ being unknown. It selects the next χ to test by selecting the maximum of the *acquisition function*, which balances exploration – improving the model in the less explored parts of the search space – and exploitation – favoring parts that the models predicts as promising. Here, we use the "Upper Confidence Bound" acquisition function (see the "information acquisition function" section below). Once the observation is made, the algorithm updates the Gaussian process to take the new data into account. In classic Bayesian optimization, the Gaussian process is initialized with a constant mean because it is assumed that all the points of the search space are equally likely to be good. The model is then progressively refined after each observation.

The key concept of the Repertoire-based Bayesian Optimization Algorithm (RBOA) is to use the output of MAP-Elites as a prior for the Bayesian optimization algorithm: thanks to the simulations, we expect some behaviors to perform better than others on the robot. To incorporate this idea into the Bayesian optimization, RBOA models the *difference* between the prediction of the repertoire and the actual performance on the real robot, instead of directly modeling the objective function. This idea is elegantly incorporated into the Gaussian process by modifying the update equation for the mean function ($\mu_t(\mathbf{x})$, equation 3):

$$\mu_t(\mathbf{x}) = \mathcal{P}(\mathbf{x}) + \mathbf{k}^\intercal \mathbf{K}^{-1}(\mathbf{P}_{1:t} - \mathcal{P}(\chi_{1:t})) \tag{4}$$

where $\mathcal{P}(\mathbf{x})$ is the performance of $\mathbf{x}$ according to the simulation and $\mathcal{P}(\chi_{1:t})$ is the performance of all the previous observations, also according to the simulation. Replacing $\mathbf{P}_{1:t}$ (eq. 3) by $\mathbf{P}_{1:t} - \mathcal{P}(\chi_{1:t})$ (eq. 4) means that the Gaussian process models the difference between the actual performance $\mathbf{P}_{1:t}$ and the performance from the repertoire $\mathcal{P}(\chi_{1:t})$. The term $\mathcal{P}(\mathbf{x})$ is the prediction of the repertoire. RBOA therefore starts with the prediction from the repertoire and corrects it with the Gaussian process.

Overall, the RBOA unites MAP-Elites, Gaussian processes, and Bayesian optimization in the following way (see top of Materials and Methods for an explanation of the notation):

procedure RBOA (REPERTOIRE-BASED BAYESIAN OPTIMIZATION ALGORITHM)
$\forall \mathbf{x} \in$ repertoire: ▷ Initialisation:
$P(f(\mathbf{x})|\mathbf{x}) = \mathcal{N}(\mu_0(\mathbf{x}), \sigma_0^2(\mathbf{x}))$ ▷ Definition of the Gaussian Process.
where
$\mu_0(\mathbf{x}) = \mathcal{P}(\mathbf{x})$ ▷ Initialize the mean prior from repertoire.
$\sigma_0^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) + \sigma_{noise}^2$ ▷ Initialize the variance prior (in the common case, $k(\mathbf{x}, \mathbf{x}) = 1$)
while $\max(\mathbf{P_{1:t}}) < \alpha \max(\mu_t(\mathbf{x}))$ **do** ▷ Iteration loop.
$\chi_{t+1} \leftarrow \arg\max_{\mathbf{x}}(\mu_t(\mathbf{x}) + \kappa\sigma_t(\mathbf{x}))$ ▷ Select next test (argmax of acquisition function).
$P_{t+1} \leftarrow$ performance(physical_robot($\mathcal{C}(\chi_{t+1})$)). ▷ Evaluation of $\mathbf{x_{t+1}}$ on the physical robot.
$P(f(\mathbf{x})|\mathbf{P}_{1:t+1}, \mathbf{x}) = \mathcal{N}(\mu_{t+1}(\mathbf{x}), \sigma_{t+1}^2(\mathbf{x}))$ ▷ Update the Gaussian Process.
where
$\mu_{t+1}(\mathbf{x}) = \mathcal{P}(\mathbf{x}) + \mathbf{k}^\intercal \mathbf{K}^{-1}(\mathbf{P}_{1:t+1} - \mathcal{P}(\chi_{1:t+1}))$ ▷ Update the mean.
$\sigma_{t+1}^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) + \sigma_{noise}^2 - \mathbf{k}^\intercal \mathbf{K}^{-1}\mathbf{k}$ ▷ Update the variance.
$$\mathbf{K} = \begin{bmatrix} k(\chi_1, \chi_1) & \cdots & k(\chi_1, \chi_{t+1}) \\ \vdots & \ddots & \vdots \\ k(\chi_{t+1}, \chi_1) & \cdots & k(\chi_{t+1}, \chi_{t+1}) \end{bmatrix} + \sigma_{noise}^2 I$$ ▷ Compute the observations' correlation matrix.
$$\mathbf{k} = \begin{bmatrix} k(\mathbf{x}, \chi_1) & k(\mathbf{x}, \chi_2) & \cdots & k(\mathbf{x}, \chi_{t+1}) \end{bmatrix}$$ ▷ Compute the $\mathbf{x}$ vs. observation correlation vector.

Table 2. Main parameters of RBOA.

σ_{noise}^2	α	ρ	κ
0.001	0.9	0.4	0.05

Kernel function The kernel function is the covariance function of the Gaussian process. It defines the influence of a controller's performance on the physical robot on the confidence in the performance estimations of not-yet-tested controllers in the repertoire that are nearby in behavior space to the tested controller (Fig. S2A).

The Squared Exponential covariance function and the Matérn kernel are the most common kernels for Gaussian processes (*24, 37, 38*). Both kernels are variants of the "bell curve". Here we chose the Matérn kernel because it is more general (it includes the Squared Exponential function as a special case) and because it allows us to control not only the distance at which effects become nearly zero (as a function of parameter ρ, Fig. S2A), but also the rate at which distance effects decrease (as a function of parameter ν).

The Matérn kernel function is computed as follows (*46, 47*) (with $\nu = 5/2$):

$$k(\mathbf{x}_1, \mathbf{x}_2) = \left(1 + \frac{\sqrt{5}d(\mathbf{x}_1, \mathbf{x}_2)}{\rho} + \frac{5d(\mathbf{x}_1, \mathbf{x}_2)^2}{3\rho^2}\right)\exp\left(-\frac{\sqrt{5}d(\mathbf{x}_1, \mathbf{x}_2)}{\rho}\right) \tag{5}$$

where $d(\mathbf{x}_1, \mathbf{x}_2)$ is the Euclidean distance in behavior space.

Because the model update step directly depends on ρ, it is one of the most critical parameters of the Intelligent Trial and Error Algorithm.

For ρ between 0.1 and 0.8, we counted the number of behaviors from the repertoire that are influenced by a single test on the real robot (we considered that a behavior was influenced when its estimated performance was affected by more than 25% of the magnitude of the update for the tested behavior). We reproduced this counting procedure for each possible behavior descriptor of a repertoire (Fig. 2): with $\rho = 0.2$, the update process does not affect any neighbor in the repertoire, with $\rho = 0.4$, it affects 10% of the behaviors, and with $\rho = 0.8$, it affects 80% of them.

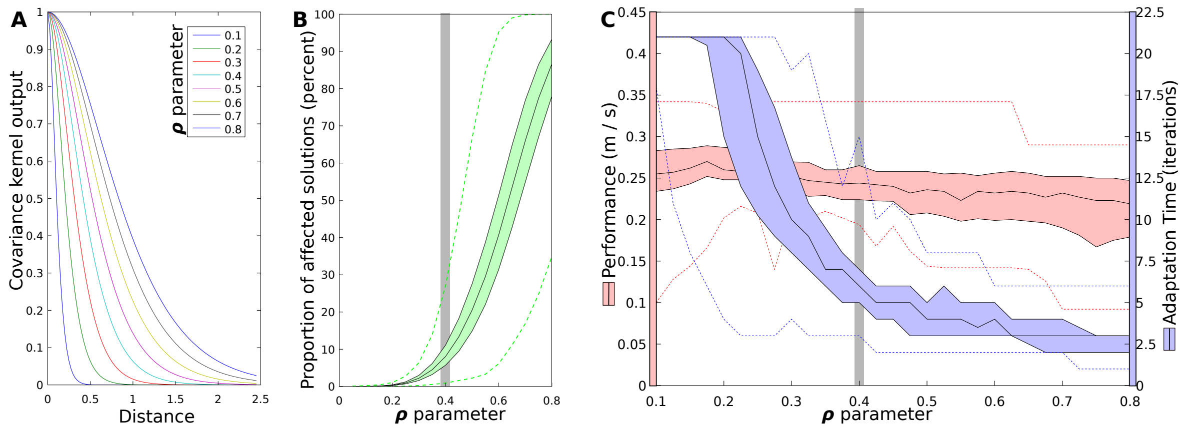

Fig. S2. (A) The shape of the Matérn kernel function for different values of parameter ρ. (B) The number of controllers in the repertoire affected by a new observation according to different values of the ρ parameter. (C) Performance and required adaptation time obtained for different values of ρ. For each ρ value, the R-BOA algorithm has been executed in simulation using 8 independently generated behavioral repertoires and with 6 different damages (each case where one leg is missing). In (B) and (C), the middle, black lines represent medians and the borders of the shaded areas show the 25^{th} and 75^{th} percentiles. The dotted lines are the minimum and maximum values. The gray bars show the ρ value chosen for the experiments in the main text.

We then repeated the experiments from the main paper with a set of possible values ($\rho \in [0.1 : 0.025 : 0.8]$) in simulation (i.e., with a simulated, damaged robot), including testing on 6 separate damage scenarios (each where the robot loses a different leg) with all 8 different behavioral repertoires. The algorithm stopped if 20 adaptation iterations passed without success according to the stopping criteria described in the main text and Section S1.3. Globally, the median performance decreases only modestly, but significantly, when the value of ρ increases: changing ρ from 0.1 to 0.8 only decreases the median value 12%, from 0.25 m/s to 0.22 m/s (p-value $= 9.3 \times 10^{-5}$ via Matlab's Wilcoxon ranksum test, Fig. S2C). The variance in performance, especially at the extreme low end of the distribution of performance values, is not constant over the range of explored values. Around $\rho = 0.3$ the minimum performance (Fig. S2C, dotted red line) is higher than the minimum performance for more extreme values of ρ.

A larger effect of changing ρ is the amount of time required to find a compensatory behavior, which decreases when the value of ρ increases (Fig. S2C). With a ρ value lower than 0.25, the algorithm rarely converges in less than the allotted 20 iterations, which occurs because many more tests are required to cover all the promising areas of the search space to know if a higher-performing behavior exists than the best-already-tested. On the other hand, with a high ρ value, the algorithm updates its predictions for the entire search space in a few observations: while fast, this strategy risks missing promising areas of the search space.

In light of these data, we chose $\rho = 0.4$ as the default value for our experiments because it represents a good trade-off between a high minimum performance and a low number of physical tests on the robot.

Information acquisition function The information acquisition function selects the next solution that will be evaluated on the physical robot. The selection is made by finding the solution that maximizes the acquisition function. This step is another optimization problem, but does not require testing the controller in simulation or reality. In general, for this optimization problem we can derive the exact equation and find a solution with gradient-based optimization (*48*). For the specific behavior space in the example problem in this paper, though, the discretized search space of the repertoire is small enough that we can exhaustively compute the acquisition value of each solution of the repertoire and then choose the maximum value.

Several different acquisition functions exist, such as the probability of improvement, the expected improvement, or the Upper Confidence Bound (UCB) (*37, 45*). We chose UCB because it provided the best results in several previous studies (*37, 45*). The equation for UCB is

$$\mathbf{x}_{t+1} = \arg\max_{\mathbf{x}} (\mu_t(\mathbf{x}) + \kappa \sigma_t(\mathbf{x})) \tag{6}$$

The acquisition function handles the exploitation/exploration trade-off of the damage recovery (RBOA) phase. In the UCB function (Eq. 6), the emphasis on exploitation vs. exploration is explicit and easy to adjust. The UCB function can be seen as the maximum value (argmax) across all solutions of the weighted sum of the expected performance (mean of the Gaussian, $\mu_t(\mathbf{x})$) and of the uncertainty (standard deviation of the Gaussian, $\sigma_t(\mathbf{x})$) of each solution. This sum is weighted by the κ factor. With a low κ, the algorithm will choose solutions that are expected to be high-performing. Conversely, with a high κ, the algorithm will focus its search on unexplored areas of the search space that may have high-performing solutions. The κ

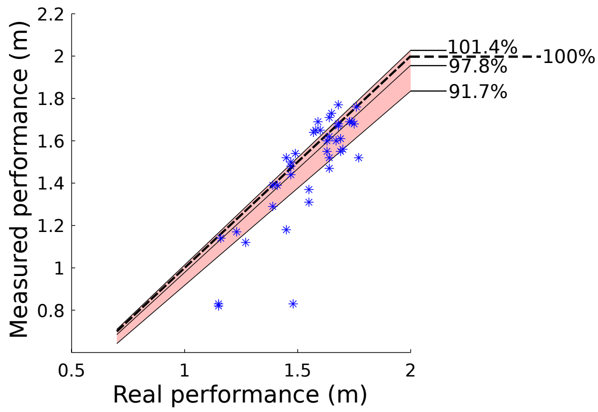

Fig. S3. Evaluation of the precision of the odometry value–the distanced traveled by the real robot–produced by the simultaneous location and mapping (SLAM) algorithm (*49, 50*). The middle solid black line represents the median deviation (in percent) and the shaded area represents the 25th and 75th percentiles of this deviation. The dashed black line indicates error-free measurements.

factor enables fine adjustments to the exploitation/exploration trade-off of the RBOA algorithm (the damage recovery step). We chose $\kappa = 0.05$. This relatively low value emphasizes exploitation over exploration. We chose this value because the exploration of the search space has already been largely performed during the behavioral repertoire creation step: the repertoire suggests which areas of the space will be high-performing, and should thus be tested, and which areas of the space are likely unprofitable, and thus should be avoided.

Stopping criterion In addition to guiding the learning process to the most promising area of the search space, the estimated performance of each solution in the repertoire also informs the algorithm of the maximum performance that can be expected on the physical robot. For example, if there is no controller in the repertoire that is expected to perform faster on the real robot than 0.3m/s, it is unlikely that a faster solution exists. This information is used in our algorithm to decide if it is worth continuing to search for a better controller; if the algorithm has already discovered a controller that performs nearly as well as the highest value predicted by the model, we can stop the search.

Formally, our stopping criterion is

$$\max(\mathbf{P}_{1:t}) \geq \alpha \max_{\mathbf{x}\in\mathcal{P}}(\mu_t(\mathbf{x})), \text{ with } \alpha = 0.9 \tag{7}$$

where $\mathbf{x}$ is a location in the discrete behavioral space (i.e. a type of behavior) and μ_t is the predicted performance of this type of behavior. Thus, when one of the tested solutions has a performance of 90% or higher of the maximum expected performance of any behavior in the repertoire, the algorithm terminates. At that point, the highest-performing solution found so far will be the compensatory behavior that the algorithm selects. An alternate way the algorithm can halt is if 20 tests on the physical robot occur without triggering the stopping criterion described in equation 7: this event only occurred in 2 of 240 experiments performed on the physical robot described in the main text. In this case, we selected the highest-performing solution encountered during the search.

Internal measure of performance In these experiments, the "mission" of the robot is to go forward as fast as possible. The performance of a controller (a set of parameters, section S1.5) is defined as how far the robot moves in a pre-specified direction in 5 seconds.

All odometry results reported on the physical robot are measured with the embedded simultaneous location and mapping (SLAM) algorithm (*51*) (section S1.4). The accuracy of this algorithm was evaluated by comparing its measurements to ones made by hand on 40 different walking gaits. These experiments revealed that the median measurement produced by the odometry algorithm is reasonably accurate, being just 2.2% lower than the handmade measurement (Fig. S3).

Some damage to the robot may make it flip over. In such cases, the odometry algorithm returns pathological distance-traveled measurements either several meters backward or forward. To remove these errors, we set all distance-traveled measurements less than zero or greater than two meters to zero. The result of this adjustment is that the algorithm appropriately considers such behaviors low-performing. Additionally, the SLAM algorithm sometimes reports substantially inaccurate low values (outliers Fig. S3). In these cases the damage recovery algorithm will assume that the behavior is low-performing and will select another working behavior, possibly a nearby behavior that was accurately measured. Thus, the overall algorithm is not substantially impacted by such infrequent under-measurements of performance.

Table 3. Parameters of the reference controller.

Leg number		0	1	2	3	4	5
First joint	α_{i_1}	1.00	1.00	1.00	1.00	1.00	1.00
	ϕ_{i_1}	0.00	0.50	0.00	0.00	0.50	0.00
	τ_{i_1}	0.5	0.5	0.5	0.5	0.5	0.5
Two last joints	α_{i_2}	0.25	0.25	0.25	0.25	0.25	0.25
	ϕ_{i_2}	0.25	0.75	0.25	0.75	0.25	0.75
	τ_{i_2}	0.5	0.5	0.5	0.5	0.5	0.5

S1.4 Physical robot

The robot is a 6-legged robot with 3 degrees of freedom (DOFs) per leg. Each DOF is actuated by position-controlled servos (MX-28 Dynamixel actuators manufactured by Robotis). The first servo controls the horizontal (front-back) orientation of the leg and the two others control its elevation. An RGB-D camera (Xtion, from ASUS) is fixed on top of the robot. Its data are used to estimate the forward displacement of the robot via an RGB-D SLAM algorithm[4] (*51*) from the robot operating system (ROS) framework[5] (*52*).

S1.5 Parametrized controller

The angular position of each DOF is governed by a periodic function γ parametrized by its amplitude α, its phase ϕ, and its duty cycle τ (the duty cycle is the proportion of one period in which the joint is in its higher position). The function is defined with a square signal of frequency 1Hz, with amplitude α, and duty cycle τ. This signal is then smoothed via a Gaussian filter in order to remove sharp transitions, and is then shifted according to the phase ϕ.

Angular positions are sent to the servos every 30 ms. In order to keep the "tibia" of each leg vertical, the control signal of the third servo is the opposite of the second one. Consequently, positions sent to the i^{th} leg are:

- $\gamma(t, \alpha_{i_1}, \phi_{i_1}, \tau_{i_1})$ for DOF 1
- $\gamma(t, \alpha_{i_2}, \phi_{i_2}, \tau_{i_2})$ for DOF 2
- $-\gamma(t, \alpha_{i_2}, \phi_{i_2}, \tau_{i_2})$ for DOF 3

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

There are 6 parameters for each leg ($\alpha_{i_1}, \alpha_{i_2}, \phi_{i_1}, \phi_{i_2}, \tau_{i_1}, \tau_{i_2}$), therefore each controller is fully described by 36 parameters. Each parameter can have one of these possible values: 0, 0.05, 0.1, ... 0.95, 1. Different values for these 36 parameters can produce numerous different gaits, from purely quadruped gaits to classic tripod gaits.

This controller is designed to be simple enough to show the performance of the algorithm in an intuitive setup. Nevertheless, the algorithm will work with any type of controller, including bio-inspired central pattern generators (*53*) and evolved neural networks (*54–57*).

S1.6 Reference controller

Our reference controller is a classic tripod gait (*2, 58–62*). It involves two tripods: legs 0-2-4 and legs 1-3-5 (Fig. S4). 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 tripod pushes the robot forward (by shifting itself backward). The lifted tripod is then placed forward in order to repeat the cycle with the other tripods. This gait is static, fast, and similar to insect gaits (*58, 63*).

Table S3 shows the 36 parameters of the reference controller. The amplitude orientation parameters (α_{i_1}) are set to 1 to produce the fastest possible gait, while the amplitude elevation parameters (α_{i_2}) are set to a small value (0.25) to keep the gait stable. The phase elevation parameters (ϕ_{i_2}) 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 (ϕ_{i_1}) are chosen by subtracting 0.25 to the phase elevation values (ϕ_{i_2}), plus a 0.5 shift for legs 3-4-5, which are on the left side of the robot. All the duty cycle parameters (τ_i) are set to 0.5 so that the motors spend the same proportion of time in their two limit positions. The actual speed of the reference controller is not important for the comparisons made in this paper: it is simply intended as a reference and to show that the performance of classic, hand-programmed gaits tend to fail when damage occurs.

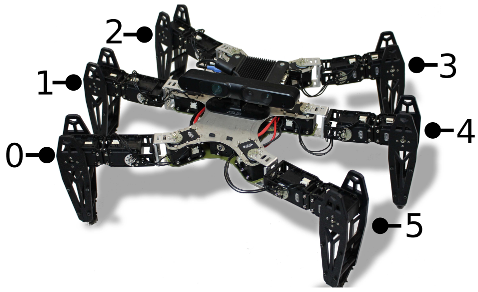

Fig. S4. The leg numbers referred to in the descriptions of robot controllers.

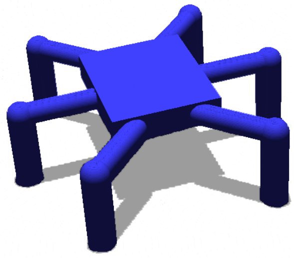

Fig. S5. Virtual robot used for the behavioral repertoire generation (MAP-Elites), simulated with the ODE dynamic simulation library (*64*) (http://www.ode.org).

S1.7 Computing hardware

All computation (on the physical robot and in simulation) was conducted on a hyperthreaded 16-core computer (Intel Xeon E5-2650 2.00GHz with 64Gb of RAM). This computational power is mainly required for the repertoire creation step. Creating one repertoire took 2 weeks, taking advantage of the fact that repertoire creation can easily be parallelized across multiple cores. Repertoire creation only needs to be performed once per robot (or robot design), and can happen before the robot is deployed. As such, the robot's onboard computer does not need to be powerful enough to create the repertoire.

The most expensive part of damage recovery is the Simultaneous Localization And Mapping (SLAM) algorithm (*49–51*), which measures the distance traveled on the physical robot. It is slow because it processes millions of 3D points per second. It can be run on less powerful computers, but doing so lowers its accuracy because fewer frames per second can be processed. As computers become faster, it should be possible to run high-accuracy SLAM algorithms in low-cost, onboard computers for robots.

The rest of the damage recovery step needs much less computational power and can easily be run on an onboard computer, such as a smartphone. That is because it takes approximately 15,000 arithmetic operations between two evaluations on the physical robot, which requires less than a second or two on current smartphones.

S1.8 Measuring how long damage recovery takes

The reported time to adapt includes the time required for the computer to select each test and the time to conduct each test on the physical robot. Overall, evaluating a controller on the physical robot takes about 8 seconds (median 8.03 seconds, 5^{th} and 95^{th} percentiles [7.95; 8.21] seconds): 0.5-1 second to initialize the robot, 5 seconds during which the robot can walk, 0.5-1 second to allow the robot to stabilize before taking the final measurement, and 1-2 seconds to run the SLAM algorithm. Identifying the first controller to test takes 0.03 [0.0216; 0.1277] seconds. The time to select the next controller to test increases depending on the number of previous experiments (Fig. S6) because the size of the Kernel Matrix (K matrix, Section S1), which is involved in many of the arithmetic operations, grows by one row and one column per test that has been conducted. For example, selecting the second test takes 0.15 [0.13; 0.22] seconds, while the 10^{th} selection takes 0.31 [0.17; 0.34] seconds.

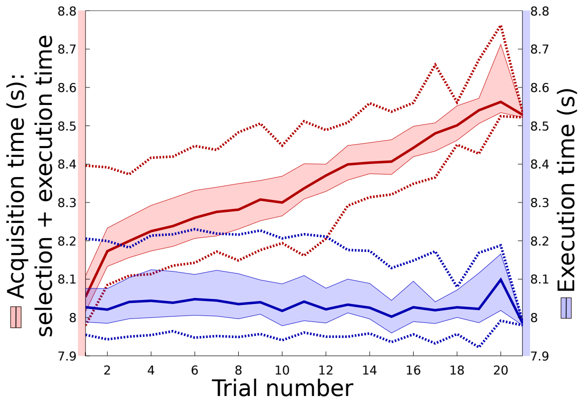

Fig. S6. Acquisition and execution time vs. trial number. The acquisition time includes the time required to select the next behavior to test from the repertoire and the time required to perform the test. These data correspond to all the experiments described in the main text (240 experiments, 6 scenarios).

Table 4. How long many previous robot learning algorithms take to run. While comparisons between these algorithms are difficult because they vary substantially in their objective, the size of the search space, and the robot they were tested on, we nonetheless can see that learning times are rarely below 20 minutes, and often take hours.

approach/article	starting behavior [★]	learning time	robot	DOFs[†]	param.[‡]	reward
Policy Gradient Methods						
Kimura et al (2001) (*65*)	n/a	80 min.	quadruped	8	72	internal
Kohl et al. (2004) (*66*)	walking	3 h	quadruped	12	12	external
Tedrake et al.(2005) (*67*)	standing	20 min.	biped	2	46	internal
Geng et al. (2006) (*68*)	n/a	4-5 min.	bipedal	4	2	internal
Christensen et al. (2013) (*69*)	n/a	10 min	quadruped	8	8	external
Evolutionary Algorithm						
Chernova et al. (2004) (*70*)	random	5 h	quadruped	12	54	external
Hornby et al. (2005) (*71*)	non-falling	25h	quadruped	19	21	internal
Barfoot et al. (2006) (*72*)	random	10 h	hexapod	12	135	external
Yosinski et al. (2011) (*54*)	random	2 h	quadruped	9	5	external
Bongard et al. (2006) (*14*)[1]	random	4 h	hexapod	12(18)	30	internal
Koos et al. (2013) (*73*)	random	20 min.	hexapod	12(18)	24	internal
Bayesian optimization						
Lizotte et al. (2007) (*36*)	center[§]	2h	quadruped	12	15	internal
Calandra et al. (2014) (*45*)	random	46 min.	biped	4	8	external
Intelligent Trial and Error (undamaged robot)	**random**	**30 sec.**	**hexapod**	**12 (18)**	**36**	**internal**
Others						
Weingarten et al. (2004) (*74*) [2]	walking	> 15 h	hexapod (Rhex)	6	8	external
Sproewitz et al. (2008) (*53*) [3]	random	60 min.	quadruped	8	5	external
Hemker et al. (2009) (*75*) [4]	walking	3-4 h	biped	24	5	external
Barfoot et al. (2006) (*72*) [5]	random	1h	hexapod	12	135	external
Erden et al. (2008) (*76*) [6]	standing	15-25min.	hexapod	18	n/a	internal

[★] Behavior used to initialize the learning algorithm.
[†] DOFs: number of controlled degrees of freedom.
[‡] param: number of learned control parameters.
[§] center: center of the search space.
[1] The authors do not provide time information, reported values come from the implementation made in Koos et al. (2013) (*73*).
[2] Nelder-Mead descent. [3] Powell method. [4] DACE (Design and Analysis of Computer Experiments).
[5] Multi-agent reinforcement learning. [6] Free-State generation with reinforcement learning.

Table 5. How long many previous robot damage recovery algorithms take to run. While comparisons between these algorithms are difficult because they vary substantially in their objective, the size of the search space, and the robot they were tested on, we nonetheless can see that damage recovery times are rarely below 20 minutes, and often take hours.

approach/article	starting behavior ⋆	learning time	robot	DOFs†	param.‡	reward
Policy Gradient Methods						
Christensen et al. (2013) (*69*)	n/a	10 min	quadruped	8	8	external
Evolutionary Algorithm						
Berenson et al. (2005) (*77*)	random	2 h	quadruped	8	36	external
Mahdavi et al. (2006) (*78*)	random	10 h	snake	12	1152	external
Bongard et al. (2006) (*14*)[1]	random	4 h	hexapod	12(18)	30	internal
Koos et al. (2013) (*73*)	random	20 min.	hexapod	12(18)	24	internal
Bayesian optimization						
Intelligent Trial and Error (damaged robot)	**random**	**1 min.**	**hexapod**	**12 (18)**	**36**	**internal**
Reinforcement learning						
Erden et al. (2008) (*76*) [2]	standing	15-25min.	hexapod	18	n/a	internal

⋆ Behavior used to initialize the learning algorithm.
† DOFs: number of controlled degrees of freedom.
‡ param: number of learned control parameters.
[1] The original authors do not provide time information, reported values come from the implementation of Koos et al. (2013) (*73*).
[2] Free-State generation with reinforcement learning.

Table 6. Knockout variants used to assess the contribution of each component of the Intelligent Trial and Error Algorithm

Variant	Behavioral repertoire creation	Priors on performance	Search algorithm	equivalent approach
Intelligent Trial and Error	MAP-Elites	yes	Bayesian Optimization	-
Variant 1	MAP-Elites	none	random search	-
Variant 2	MAP-Elites	none	Bayesian optimization	-
Variant 3	MAP-Elites	none	policy gradient	-
Variant 4	none	none	Bayesian optimization	Lizotte et al. (2007) (*36*)
Variant 5	none	none	policy gradient	Kohl et al. (2004) (*66*)

Supplementary Experiments S1

The contribution of each subcomponent of the Intelligent Trial and Error Algorithm

The Intelligent Trial and Error Algorithm relies on three main concepts: (1) the creation of a behavioral repertoire in simulation via the MAP-Elites algorithm, (2) searching this repertoire with a Bayesian optimization algorithm to find behaviors that perform well on the physical robot, and (3) initializing this Bayesian optimization search with the performance predictions obtained via the MAP-Elites algorithm: note that the second step could be performed without the third step by searching through the MAP-Elites-generated behavioral repertoire with Bayesian optimization, but having the initial priors uniformly set to the same value. We investigated the contribution of each of these subcomponents by testing five variants of our algorithm (Table S6): in each of them, we deactivated one of these three subcomponents or replaced it with an alternative algorithm from the literature. The variants are as follows:

- Variant 1 (MAP-Elites in 6 dimensions + random search): evaluates the benefit of searching the repertoire via Bayesian optimization by searching that repertoire with random search instead. Each iteration, a behavior is randomly selected from the repertoire and tested on the robot. The best one is kept.
- Variant 2 (MAP-Elites in 6 dimensions + Bayesian optimization, no use of priors): evaluates the contribution of initializing the Gaussian process with the performance predictions of the behavioral repertoire. In this variant, the Gaussian process is initialized with a constant mean (the average performance of the repertoire: 0.24 m/s) at each location in the behavior space and a constant variance (the average variance of the repertoire's performance: 0.005 m^2/s^2).
- Variant 3 (MAP-Elites in 6 dimensions + policy gradient): evaluates the benefit of Bayesian optimization compared to a more classic, local search algorithm (*22, 66*); there is no obvious way to use priors in policy gradient algorithms.
- Variant 4 (Bayesian optimization in the original parameter space of 36 dimensions): evaluates the contribution of using a repertoire in a lower-dimensional behavioral space. This variant searches directly in the original 36-dimensional parameter space instead of reducing that space to the lower-dimensional (six-dimensional) behavior space. Thus, in this variant no repertoire of behaviors is produced ahead of time: the algorithm searches directly in the original, high-dimensional space. This variant corresponds to one of the best algorithms known to learn locomotion patterns (*36, 45*). In this variant, the Gaussian process is initialized with a constant mean set to zero and with a constant variance (0.002m^2/s^2).
- Variant 5 (Policy gradient in the original parameter space of 36 dimensions): a stochastic gradient descent in the original parameter space (*66*). This approach is a classic reinforcement learning algorithm for locomotion (*22*) and it is a baseline in many papers (e.g. (*36*)).

It was necessary to compare these variants in simulation because doing so on the physical robot would have required months of experiments and would have repeatedly worn out or broken the robot. We modified the simulator from the main experiments (Section S1.2) to emulate 6 different possible damage conditions, each of which involved removing a different leg. For variants in which MAP-Elites creates a repertoire (variants 1, 2 and 3), we used the same repertoires from the main experiments (the eight independently generated repertoires, which were all generated with a simulation of the undamaged robot): In these cases, we launched ten replicates of each variant for each of the eight repertoires and each of the six damage conditions. There are therefore $10 \times 8 \times 6 = 480$ replicates for each of those variants. For the other variants (4 and 5), we replicated each experiment 80 times for each of the six damage conditions, which also led to $80 \times 6 = 480$ replicates per variant. In all these simulated experiments, to roughly simulate the distribution of noisy odometry measurements on the real robot, the simulated performance values were randomly perturbed with a multiplicative Gaussian noise centered on 0.95 with a standard deviation of 0.1.

We analyze the fastest walking speed achieved with each variant after two different numbers of trials: the first case is after 16 trials, which was the maximum number of iterations used by the Intelligent Trial and Error Algorithm (Fig. 3, maximum 16 trials, median 5 trials), and the second case is after 150 trials, which is approximately the number of trials used in previous work (Table 4, (*36, 45, 66*)).

Results

After 16 trials on the robot, Intelligent Trial and Error significantly outperforms all the variants (Fig. S7B, $p < 10^{-60}$, median Intelligent Trial and Error performance: 0.25 [0.20; 0.30] m/s), demonstrating that the three main components of the algorithm are needed to quickly find high-performing behaviors. Among the investigated variants, the random search in the repertoire performs the best (Variant 1: 0.21 [0.17; 0.26] m/s), followed by Bayesian optimization in the repertoire (Variant 2: 0.17 [0.12; 0.21] m/s), and policy gradient in the repertoire (Variant 3: 0.13 [0; 0.24] m/s). Variants that search directly in the parameter space did not find any working behavior (Variant 4, Bayesian optimization: 0.03m/s, [0.01; 0.08]; Variant 5, policy gradient: 0.02 [0; 0.06] m/s).

There are two reasons that random search performs better than one might expect. First, the repertoire only contains high-performing solutions, which are the result of the intense search of the MAP-Elites algorithm (20 million evaluations in simulation). The repertoire thus already contains high-performing gaits of nearly every possible type. Therefore, this variant is not testing random controllers, but is randomly selecting high-performing solutions. Second, Bayesian optimization and policy gradient are not designed for such a low number of trials: without the priors on performance predictions introduced in the Intelligent Trial and Error Algorithm, the Bayesian optimization process needs to learn the overall shape of the search space to model it with a Gaussian process. 16 trials is too low a number to effectively sample six dimensions (for a uniform sampling with only two possible values in each dimension, $2^6 = 64$ trials are needed; for five possible values, $5^6 = 15,625$ samples are needed). As a consequence, with this low number of trials, the Gaussian process that models the performance function is not informed enough to effectively guide the search. For the policy gradient algorithm, a gradient is estimated by empirically measuring the partial derivative of the performance function in each dimension. To do so, following (*66*), the policy gradient algorithm performs 15 trials at each iteration. Consequently, when only 16 trials are allowed, it iterates only once. In addition, policy gradient is a local optimization algorithm that highly depends on the starting point (which is here chosen randomly), as illustrated by the high variability in the performance achieved with this variant (Fig. S7B).

The issues faced by Bayesian optimization and policy gradient are exacerbated when the algorithms search directly in the original, 36-dimensional parameter space instead of the lower-dimensional (six-dimensional) behavior space of the repertoire. As mentioned previously, no working controller was found in the two variants directly searching in this high-dimensional space.

Overall, the analysis after 16 trials shows that:

- The most critical component of the Intelligent Trial and Error Algorithm is the MAP-Elites algorithm, which reduces the search space and produces a repertoire of high-performing behaviors in that space: $p < 5 \times 10^{-64}$ when comparing variants searching in the behavioral repertoire space vs. variants that search in the original, higher-dimensional space of motor parameters.
- Bayesian optimization critically improves the search, but only when it is initialized with the performance obtained in simulation during the behavioral repertoire creation step (with initialization: 0.25 [0.20; 0.30] m/s, without initialization: 0.17 [0.12; 0.21] m/s, $p = 10^{-148}$).

To check whether these variants might perform better if allowed the number of physical evaluations typically given to previous state-of-the-art algorithms (Table S4, (*36, 45, 66*)), we continued the experiments until 150 trials on the robot were conducted (Fig. S7C). Although Intelligent Trial and Error still outperforms all the variants ($p < 10^{-15}$), the difference is reduced (Intelligent Trial and Error: 0.27 [0.22; 0.33] m/s, random search: 0.25 [0.22; 0.30] m/s, Bayesian optimization: 0.23 [0.19; 0.28] m/s, policy search: 0.23 [0.19, 0.29] m/s). These results are consistent with the previously published results (*22, 36, 45, 66*), which optimize in 4 to 10 dimensions in a few hundred trials. Nevertheless, when MAP-Elites is not used, i.e. when we run these algorithms in the original 36 dimensions for 150 physical evaluations, Bayesian optimization and policy gradient both perform much worse (Bayesian optimization: 0.08 [0.05; 0.12]; policy gradient: 0.07 [0.02; 0.12] m/s). These results shows that MAP-Elites is a powerful method to reduce the dimensionality of a search space for learning algorithms, in addition to providing helpful priors about the search space that speed up Bayesian optimization.

Overall, these additional experiments demonstrate that each of the three main components of the Intelligent Trial and Error Algorithm substantially improves performance. The results also indicate that Intelligent Trial and Error significantly outperforms previous algorithms and can be considered the state of the art.

Supplementary Experiments S2

Robustness to environmental changes

The repertoire creation algorithm (MAP-Elites) uses an undamaged robot on flat terrain. The main experiments show that this algorithm provides useful priors for damage recovery on a flat terrain. In these supplementary experiments, we evaluated, in simulation, if the repertoire created on flat terrain also provides a useful starting point for discovering gaits for sloped terrains.

We first evaluated the effect slopes have on undamaged robots (Fig. S8A). We launched 10 replicates for each of the eight repertoires and each one-degree increment between $-20°$ and $+20°$, for a total of $10 \times 8 \times 41 = 3280$ experiments. As in Supplementary Experiments S1, to roughly simulate the distribution of noisy odometry measurements on the real robot, we perturbed performance values with a multiplicative Gaussian noise centered on 0.95 with a standard deviation of 0.1.

The results show that, when the slope is negative (descending), the Intelligent Trial and Error approach finds fast gaits in fewer than than 3 trials. For reference, a hand-designed, classic, tripod gait (section S1.5) falls on slopes below $-15°$ degrees. When the slope is positive (ascent), Intelligent Trial and Error finds slower behaviors, as is expected, but even above $10°$ the gait learned by Intelligent Trial and Error outperforms the reference gait on flat ground. Overall, for every slope angle, the controller found by Intelligent Trial and Error is faster than the hand-designed reference controller.

We further evaluated damage recovery performance for these same slopes. We used the same setup as Experiments S1 (6 damage conditions). We launched 10 replicates for each damage condition, for each archive, and each two-degree increment between $-20°$ and $+20°$ degrees. There are therefore 480 replicates for each degree between $-20°$ and $+20°$, for a total of $480 \times 21 = 10080$ experiments.

The results show that Intelligent Trial and Error is not critically affected by variations of slope between $-10°$ and $+10°$ (Fig. S8B): for these slopes, and for all 6 damage conditions, Intelligent Trial and Error finds fast gaits (above 0.2 m/s) in less than 15 physical tests on the robot despite the slope. As expected, it finds faster gaits for negative slopes (descent) and slower gaits for positive slopes (ascent). For slopes below $-10°$ and above $10°$, the algorithm performs worse and requires more trials. These results likely are caused by the constraints placed on the controller and the limited sensors on the robot, rather than the inabilities of the algorithm. Specifically, the controller was kept simple to make the science clearer, more intuitive, and more reproducible. Those constraints, of course, prevent it from performing the more complex behaviors necessary to deal with highly sloped terrain. For example, the constraints prevent the robot from keeping its legs vertical on sloped ground, which would substantially reduce slippage. Nevertheless, the median Intelligent Trial and Error compensatory gait still outperforms the median performance of the reference gait on all slope angles.

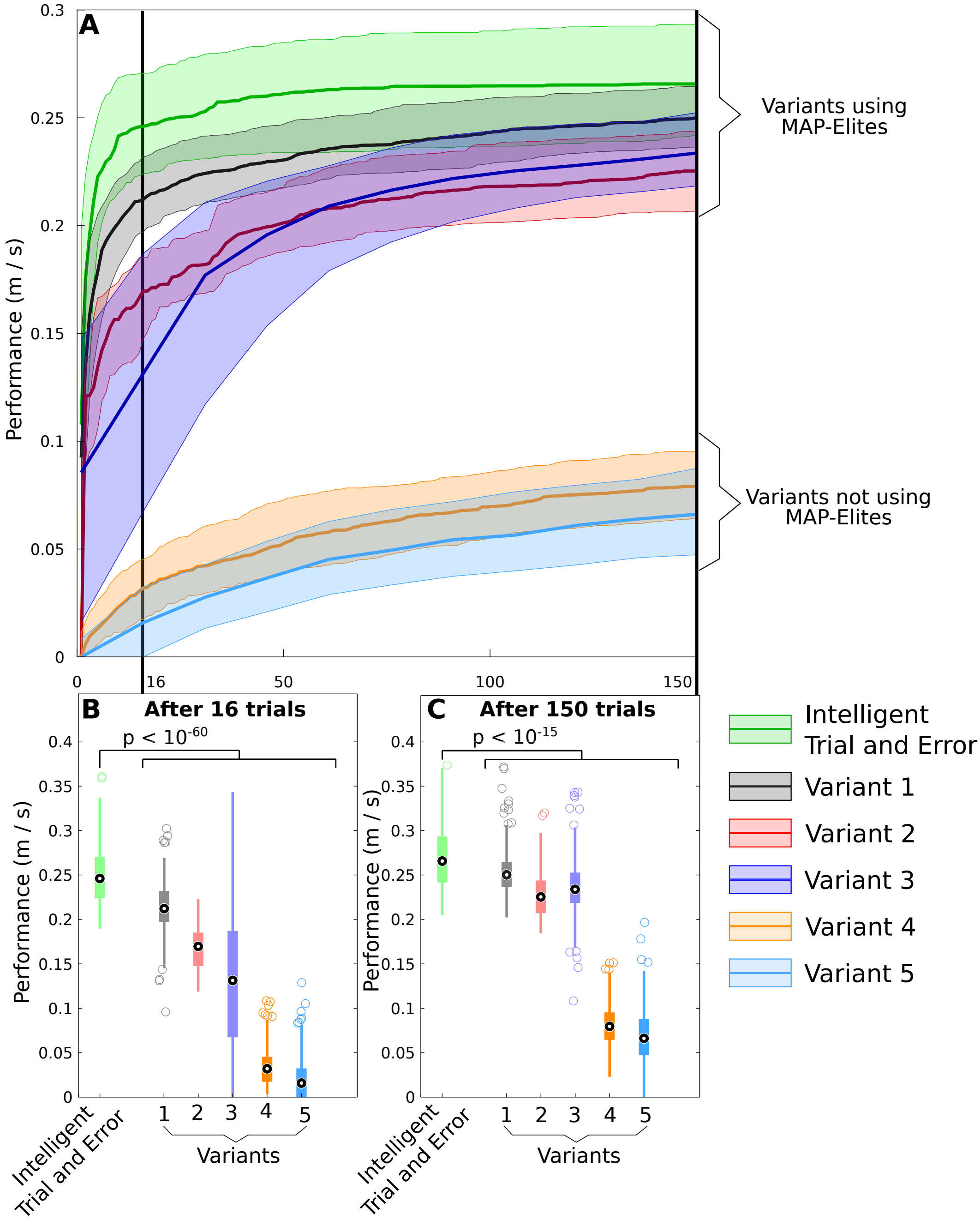

Fig. S7. (A) The walking speed achieved with Intelligent Trial and Error and several "knockout" variants that are missing one of the algorithm's key components (the variants are summarized in Table S6). Some (Variants 4 and 5) correspond to state-of-the-art learning algorithms (*36, 66*). The bold lines represent the medians and the colored areas extend to the 25th and 75th percentiles. (B, C) The speed of the compensatory behavior discovered by each algorithm after 16 and 150 physical evaluations on the robot, respectively.

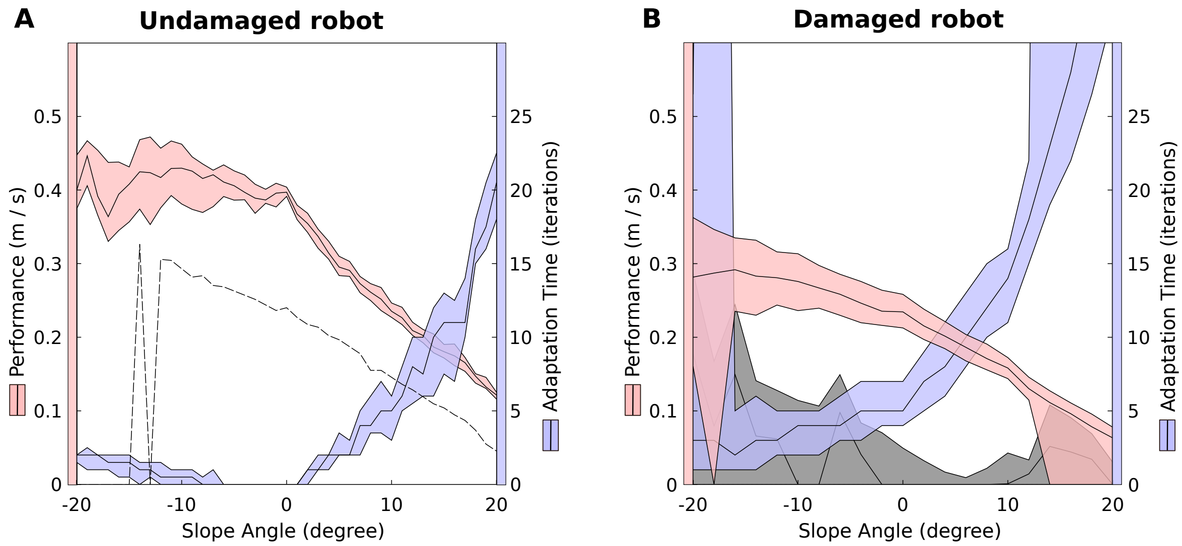

Fig. S8. Performance and required adaptation time for Intelligent Trial and Error when the robot must adapt to walk on differently sloped terrain. (A) For an undamaged robot, on all slope angles, with very few physical trials, the Intelligent Trial and Error Algorithm (pink shaded region) finds fast gaits that outperform the reference gait (black dotted line). (B) On damaged robots, the median compensatory behavior found via Intelligent Trial and Error outperforms the median reference controller on all slope angles. The middle, black lines represent medians, while the colored areas extend to the 25th and 75th percentiles. In (A), the black dashed line is the performance of a classic tripod gait for reference. In (B), the reference gait is tried in all six damage conditions and its median (black line) and 25th and 75th percentiles (black colored area) are shown.

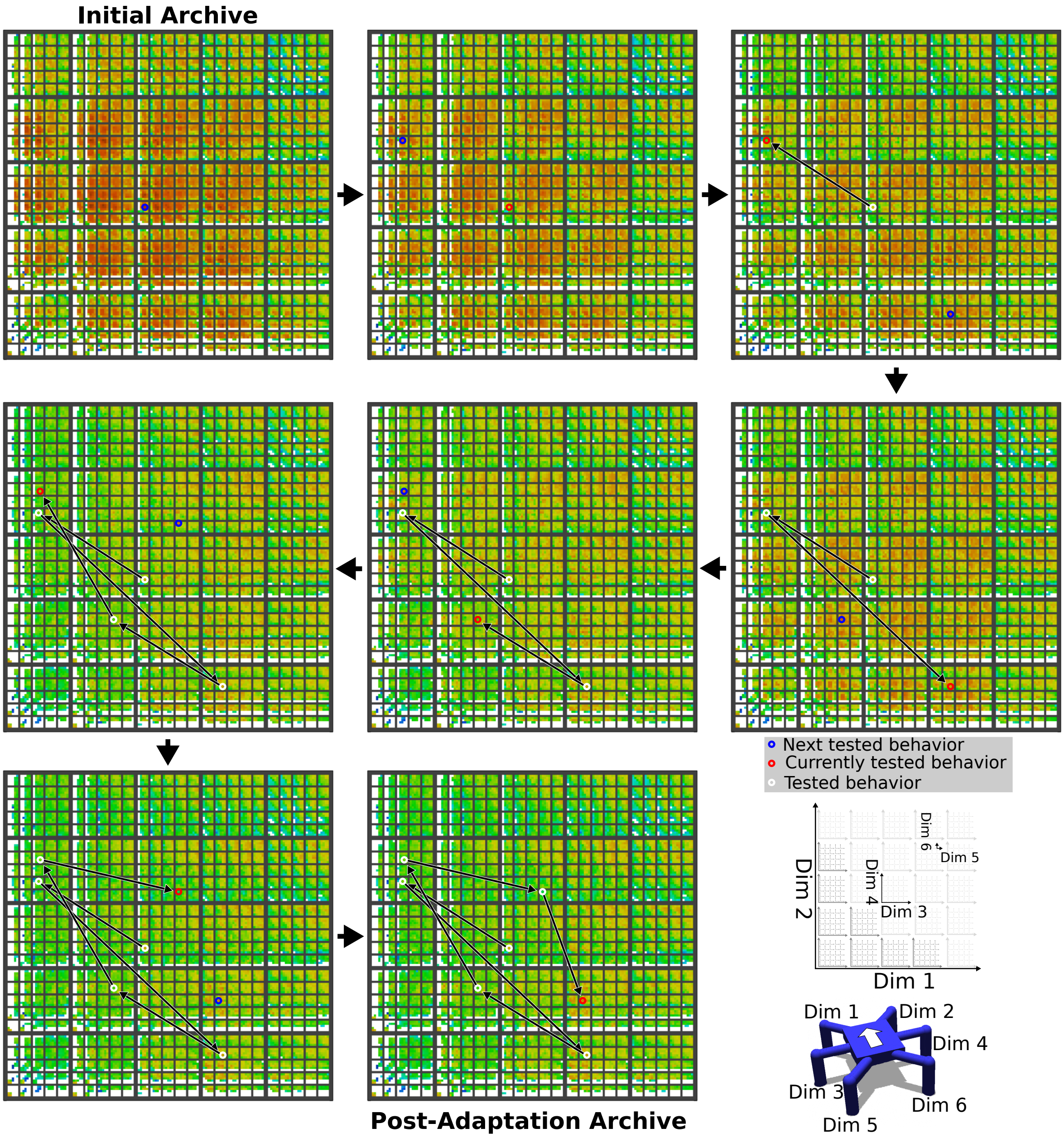

Fig. S9. Behavioral repertoire states during the adaptation process. In this scenario, the robot loses the leg number 4 (Fig. S4).

Supplementary References

35. A. E. Eiben, J. E. Smith, *Introduction to evolutionary computing* (Springer, 2003).
36. D. J. Lizotte, T. Wang, M. H. Bowling, D. Schuurmans, *Proceedings of the the International Joint Conference on Artificial Intelligence (IJCAI)* (2007), vol. 7, pp. 944–949.
37. E. Brochu, V. M. Cora, N. De Freitas, *arXiv preprint arXiv:1012.2599* (2010).
38. J. Snoek, H. Larochelle, R. P. Adams, *Advances in Neural Information Processing Systems 25 (NIPS)* (2012), pp. 2951–2959.
39. A. J. Booker, *et al.*, *Structural optimization* **17**, 1 (1999).
40. A. I. J. Forrester, A. J. Keane, *Progress in Aerospace Sciences* **45**, 50 (2009).
41. Y. Jin, *Swarm and Evolutionary Computation* **1**, 61 (2011).
42. T. W. Simpson, T. M. Mauery, J. J. Korte, F. Mistree, *American Institute of Aeronautics and Astronautics* **98**, 1 (1998).
43. D. R. Jones, M. Schonlau, W. J. Welch, *Journal of Global optimization* **13**, 455 (1998).
44. J. Sacks, W. J. Welch, T. J. Mitchell, H. P. Wynn, *et al.*, *Statistical science* **4**, 409 (1989).
45. R. Calandra, A. Seyfarth, J. Peters, M. P. Deisenroth, *Proceedings of 2014 IEEE International Conference on Robotics and Automation (ICRA)* (2014).
46. B. Matérn, *et al.*, *Meddelanden fran statens Skogsforskningsinstitut* **49** (1960).
47. M. L. Stein, *Interpolation of spatial data: some theory for kriging* (Springer, 1999).
48. A. V. Fiacco, G. P. McCormick, *Nonlinear programming: sequential unconstrained minimization techniques*, vol. 4 (Siam, 1990).
49. S. Thrun, W. Burgard, D. Fox, *et al.*, *Probabilistic robotics* (MIT press Cambridge, 2005).
50. M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, M. Csorba, *IEEE Transactions on Robotics and Automation* **17**, 229 (2001).
51. I. Dryanovski, R. G. Valenti, J. Xiao, *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2013), pp. 2305–2310.
52. M. Quigley, *et al.*, *Proceedings of ICRA's workshop on Open Source Software* (2009).
53. A. Sproewitz, R. Moeckel, J. Maye, A. Ijspeert, *The International Journal of Robotics Research* **27**, 423 (2008).
54. J. Yosinski, *et al.*, *Proceedings of ECAL* pp. 890–897 (2011).
55. J. Clune, K. Stanley, R. Pennock, C. Ofria, *IEEE Transactions on Evolutionary Computation* **15**, 346 (2011).
56. J. Clune, B. Beckmann, C. Ofria, R. Pennock, *Proceedings of the IEEE Congress on Evolutionary Computation* (2009), pp. 2764–2771.
57. S. Lee, J. Yosinski, K. Glette, H. Lipson, J. Clune, *Applications of Evolutionary Computing* (Springer, 2013).
58. D. Wilson, *Annual Review of Entomology* **11**, 103 (1966).
59. U. Saranli, M. Buehler, D. Koditschek, *The International Journal of Robotics Research* **20**, 616 (2001).
60. J. Schmitz, J. Dean, T. Kindermann, M. Schumm, H. Cruse, *The biological bulletin* **200**, 195 (2001).
61. X. Ding, Z. Wang, A. Rovetta, J. Zhu, *Proceedings of Conference on Climbing and Walking Robots (CLAWAR)* pp. 291–310 (2010).
62. S. Steingrube, M. Timme, F. Wörgötter, P. Manoonpong, *Nature Physics* **6**, 224 (2010).
63. F. Delcomyn, *Journal of Experimental Biology* **54**, 443 (1971).
64. A. Boeing, T. Bräunl, *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia* (ACM, 2007), pp. 281–288.
65. H. Kimura, T. Yamashita, S. Kobayashi, *Proceedings of IEEE Conference on Decision and Control (CDC)* (IEEE, 2001), vol. 1, pp. 411–416.
66. N. Kohl, P. Stone, *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2004), vol. 3, pp. 2619–2624.
67. R. Tedrake, T. Zhang, H. Seung, *Proceedings of Yale workshop on Adaptive and Learning Systems* (2005).
68. T. Geng, B. Porr, F. Wörgötter, *The International Journal of Robotics Research* **25**, 243 (2006).
69. D. J. Christensen, J. C. Larsen, K. Stoy, *Evolving Systems* pp. 1–12 (2013).
70. S. Chernova, M. Veloso, *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2004), vol. 3, pp. 2562–2567.
71. G. Hornby, S. Takamura, T. Yamamoto, M. Fujita, *IEEE Transactions on Robotics* **21**, 402 (2005).
72. T. Barfoot, E. Earon, G. D'Eleuterio, *Robotics and Autonomous Systems* **54**, 864 (2006).
73. S. Koos, A. Cully, J.-B. Mouret, *The International Journal of Robotics Research* **32**, 1700 (2013).
74. J. Weingarten, G. Lopes, M. Buehler, R. Groff, D. Koditschek, *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (2004), vol. 3, pp. 2153–2158.
75. T. Hemker, M. Stelzer, O. Von Stryk, H. Sakamoto, *The International Journal of Robotics Research* **28**, 303 (2009).
76. M. S. Erden, K. Leblebicioğlu, *Robotics and Autonomous Systems* **56**, 199 (2008).
77. D. Berenson, N. Estevez, H. Lipson, *Proceedings of NASA/DoD Conference on Evolvable Hardware* (2005), pp. 12–19.
78. S. Mahdavi, P. Bentley, *Autonomous Robots* **20**, 149 (2006).

S2 Caption for Supplementary Videos

S2.1 Video S1

This video can be viewed at:
http://www.youtube.com/watch?v=p6oegqDrCDs&index=2&list=PLc7kzd2NKtSdvG2OXKeM9CnfRG9yPG6Sb

Damage Recovery in Robots via Intelligent Trial and Error. The video shows the Intelligent Trial and Error Algorithm in action for two different damage conditions: a leg that has lost power (Fig. 3C3) and a broken leg (Fig. 3C2). Initially, when the robot is undamaged, a hand-designed, classic tripod gait (section S1.6, (*2*)) performs well. Once damage occurs, however, this reference gait no longer works. The Intelligent Trial and Error Algorithm is initiated and quickly finds fast, compensatory behaviors for both damage conditions.

S2.2 Video S2

This video can be viewed at:
http://www.youtube.com/watch?v=ycLspV5lXK8&list=PLc7kzd2NKtSdvG2OXKeM9CnfRG9yPG6Sb&index=2

A Behavior Repertoire Containing Many Different Types of Walking Gaits. In the behavioral repertoire creation step, the MAP-Elites algorithm produces a collection of different types of walking gaits. The video shows several examples of the different types of behaviors that are produced, from classic hexapod gaits to more unexpected forms of locomotion.

S3 Source code

The source code (for GNU/Linux) for the experiments of this paper is available at the following URL:
http://pages.isir.upmc.fr/~mouret/code/ite_source_code.tar.gz