



**HAL**  
open science

## Robots that can adapt like animals

Antoine Cully, Jeff Clune, Danesh Tarapore, Jean-Baptiste Mouret

► **To cite this version:**

Antoine Cully, Jeff Clune, Danesh Tarapore, Jean-Baptiste Mouret. Robots that can adapt like animals. *Nature*, 2015, 521 (7553), pp.503-507. 10.1038/nature14422 . hal-01158243

**HAL Id: hal-01158243**

**<https://hal.science/hal-01158243>**

Submitted on 30 May 2015

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Robots that can adapt like animals

Antoine Cully,<sup>1,2</sup> Jeff Clune,<sup>6</sup> Danesh Tarapore,<sup>1,2</sup> Jean-Baptiste Mouret<sup>1-5,\*</sup>

As robots leave the controlled environments of factories to autonomously function in more complex, natural environments<sup>1,2,3</sup>, they will have to respond to the inevitable fact that they will become damaged<sup>4,5</sup>. However, 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 limited to their pre-specified self-sensing abilities, can diagnose only anticipated failure modes<sup>6</sup>, and require a pre-programmed contingency plan for every type of potential damage, an impracticality for complex robots<sup>4,5</sup>. Here we introduce an intelligent trial and error algorithm that allows robots to adapt to damage in less than two minutes, without requiring self-diagnosis or pre-specified contingency plans. Before deployment, a robot exploits a novel algorithm to create a detailed map of the space of high-performing behaviors: This map represents the robot’s intuitions about what behaviors it can perform and their value. If the robot is damaged, it uses these intuitions to guide a trial-and-error learning algorithm that conducts intelligent experiments to rapidly discover a compensatory behavior that works in spite of the damage. Experiments reveal successful adaptations for a legged robot injured in five different ways, including damaged, broken, and missing legs, and for a robotic arm with joints broken in 14 different ways. 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<sup>7</sup>, and have the power to deliver tremendous benefits to society, such as in search and rescue<sup>8</sup>, disaster response<sup>9</sup>, health care<sup>3</sup>, and transportation<sup>10</sup>. They are also invaluable tools for scientific exploration, whether of distant planets<sup>1,4</sup> or deep oceans<sup>2</sup>. A major obstacle to their widespread adoption in more complex environments outside of factories is their fragility<sup>4,5</sup>: 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<sup>11,12,13,14</sup>. Such self-diagnosing robots are expensive, because self-monitoring sensors are expensive, and are difficult to design, because robot engineers cannot foresee every possible situation: this approach often fails either because the diagnosis is incorrect<sup>12,13</sup> or because an appropriate contingency plan is not provided<sup>14</sup>.

Injured animals respond differently: they learn by trial and error how to compensate for damage (e.g. learning which limp minimizes pain)<sup>15,16</sup>. Similarly, trial-and-error learning algorithms could allow robots to creatively discover compensatory behaviors 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”<sup>17</sup>: the fastest

algorithms constrain the search to a few behaviors (e.g. tuning only 2 parameters, requiring 5-10 minutes) or require human demonstrations<sup>17</sup>. Algorithms without these limitations take several hours<sup>17</sup>. 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.

Here, we show that rapid adaptation can be achieved by guiding an intelligent trial-and-error learning algorithm with an automatically generated, pre-computed, behavior-performance map that predicts the performance of thousands of different behaviors (Supplementary Video S1). The key insight is that, whereas current learning algorithms either start with no knowledge of the search space<sup>17</sup> or with minimal knowledge from a few human demonstrations<sup>17,18</sup>, animals better understand the space of possible behaviors and their value from previous experience<sup>19</sup>, enabling injured animals to intelligently select tests that validate or invalidate whole families of promising compensatory behaviors.

We have robots store knowledge from previous experience in the form of a map of the behavior-performance space. Guided by this map, a damaged robot tries different types of behaviors that are predicted to perform well and, as tests are conducted, updates its estimates of the performance of those types of behaviors. The process ends when the robot predicts that the most effective behavior has already been discovered. The result is a robot that quickly discovers a way to compensate for damage (e.g. Fig. 1C) without a detailed mechanistic understanding of its cause, as occurs with animals. We call this approach “Intelligent Trial and Error” (Fig. 1D).

The behavior-performance map is created with a novel algorithm and a simulation of the robot, which either can be a standard physics simulator or can be automatically discovered<sup>12</sup>. The robot’s designers only have 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 a gait (a behavioral measure) and speed (a performance measure). For grasping, performance could be the amount of surface contact, and it has been demonstrated that 90% of effective poses for the 21-degree-of-freedom human hand can be captured by a 3-dimensional behavioral space<sup>20</sup>. To fill in the behavior-performance map, an optimization algorithm simultaneously searches for a high-performing solution at each point in the behavioral space (Fig. 2A,B and Extended Data Fig. 1). This step requires simulating millions of behaviors, but needs to be performed only once per robot design before deployment (Methods).

A low confidence is assigned to the predicted performance of behaviors stored in this behavior-performance map because they have not been tried in reality (Fig. 2B and Extended Data Fig. 1). During the robot’s mission, if it senses a performance drop, it selects the most promising behavior from the behavior-performance map, tests it, and measures its performance. The robot subsequently updates its prediction for that behavior and nearby behaviors, assigns high confidence to these predictions (Fig. 2C and Extended Data Fig. 1), and continues the selection/test/update process until it finds a satisfactory compensatory behavior (Fig. 2D and Extended Data Fig. 1).

All of these ideas are technically captured via a Gaussian process model<sup>21</sup>, which approximates the performance function

<sup>1</sup> Sorbonne Universités, UPMC Univ Paris 06, UMR 7222, ISIR, F-75005, Paris

<sup>2</sup> CNRS, UMR 7222, ISIR, F-75005, Paris, France

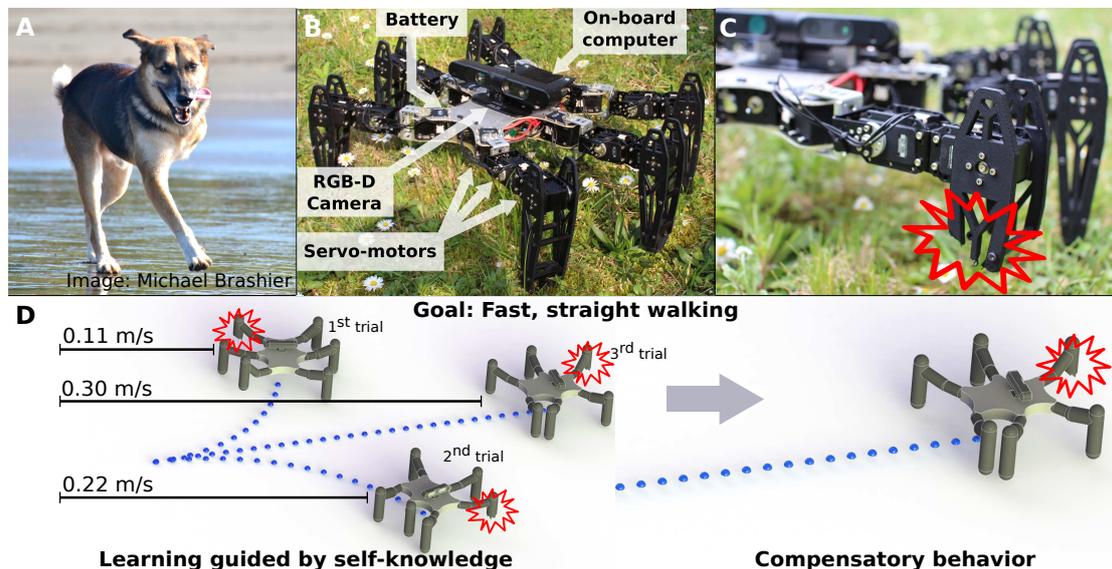
<sup>3</sup> Inria, Villers-lès-Nancy, F-54600, France

<sup>4</sup> CNRS, Loria, UMR 7503, Vandœuvre-lès-Nancy, F-54500, France

<sup>5</sup> Université de Lorraine, Loria, UMR 7503, Vandœuvre-lès-Nancy, F-54500, France

<sup>6</sup> University of Wyoming, Laramie, WY, USA

\* Corresponding author: [jean-baptiste.mouret@inria.fr](mailto:jean-baptiste.mouret@inria.fr)



**Figure 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 hexapod may have 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 map of the behavior-performance space. 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.

with already acquired data, and a Bayesian optimization procedure<sup>22,23</sup>, which exploits this model to search for the maximum of the performance function (Methods). 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) (Methods). 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 (Methods). 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 behavior-performance map (Methods).

We first 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 (Supplementary Methods). The gait is parametrized by 36 real-valued parameters that describe the amplitude of oscillation, phase shift, and duty cycle for each joint (Supplementary Methods). The behavior space is 6-dimensional, where each dimension is the proportion of time the  $i^{\text{th}}$  leg spends in contact with the ground (i.e. the duty factor)<sup>7</sup> (Supplementary Methods).

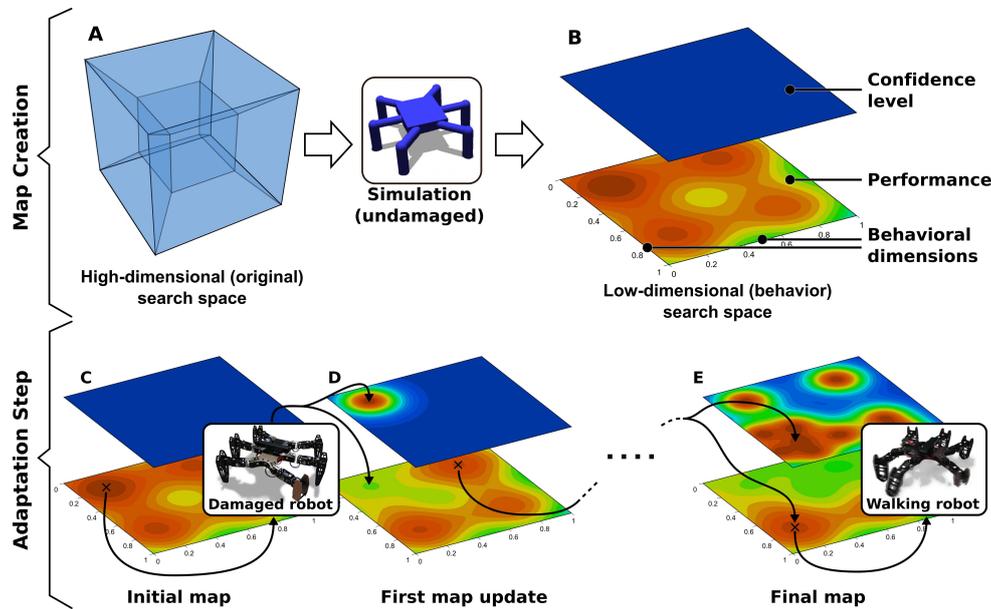
The created behavior-performance map contains approximately 13,000 different gaits (Supplementary Video S2 shows examples). We tested our robot in six different conditions: undamaged (Fig. 3A:C1), four different structural failures (Fig. 3A:C2-C5), and a temporary leg repair (Fig. 3A:C6). We compare the walking speed of resultant gaits with a widely-used, classic, hand-designed tripod gait<sup>7</sup> (Supplementary Methods). For each of the 6 damage conditions, we ran our adaptation step 5 times for each of 8 independently generated behavior-performance maps (with the default “duty factor” behavioral description), leading to  $6 \times 5 \times 8 = 240$  experiments in total. We also ran our adaptation step 5 times on 8 independently generated behavior-performance maps defined by an alternate behavioral descrip-

tion (“body orientation”, see Supplementary Methods) on two damage conditions (Fig. 3B-C), leading to  $2 \times 5 \times 8 = 80$  additional experiments.

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  $0.32 \text{ m/s}$ , 5<sup>th</sup> and 95<sup>th</sup> percentiles [0.26; 0.36] vs.  $0.24 \text{ m/s}$ ), 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 ( $\sim 0.04 \text{ m/s}$  for the four damage conditions, Fig. 3B:C2-C5). After Intelligent Trial and Error, the compensatory gaits achieve a reasonably fast speed ( $> 0.15 \text{ m/s}$ ) and are between 3 and 7 times more efficient than the reference gait for that damage condition (in  $\text{m/s}$ , 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] vs  $0.12$ ).

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 (Extended Data Fig. 2), and that Intelligent Trial and Error can help with another major challenge in robotics: adapting to new environments (Extended Data Fig. 3). On the undamaged or repaired robot (Fig. 3: C6), Intelligent Trial and Error learns a walking gait in less than 30 seconds (Fig. 3C, undamaged: 24 [16; 41] seconds, 3 [2; 5] physical trials, repaired: 29 [16; 82] seconds, 3.5 [2; 10] trials). For the four damage scenarios, the robot adapts in approximately one minute (66 [24; 134] seconds, 8 [3; 16] trials). Our results are qualitatively unchanged when using different behavioral characterizations, including randomly choosing 6 descriptors among 63 possibilities (Fig. 3B-C and Extended Data Fig. 4). Additional experiments show that reducing the high-dimensional parameter space to a low-dimensional behavior space via the behavior-performance map is the key component for intelligent trial and error: standard Bayesian optimization in the original parameter space does not find working controllers (Extended Data Fig. 2).

We investigated how the behavior-performance map is updated when the robot loses a leg (Fig. 3A:C4). Initially the map



**Figure 2 | (A & B). Creating the behavior-performance map:** 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 then automatically searched to find a high-performing behavior at each point in the low-dimensional behavior space, creating a “behavior-performance” map of the performance potential of each location in the low-dimensional space. In our hexapod robot experiments, the behavior space is six-dimensional: the portion of time that each leg is in contact with the ground. The confidence regarding the accuracy of the predicted performance for each behavior in the behavior-performance map is initially low because no tests on the physical robot have been conducted. **(C & D) Adaptation Step:** After damage, the robot selects a promising behavior, tests it, updates the predicted performance of that behavior in the behavior-performance map, 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 accordingly. 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 behavior-performance map, a value that can decrease with each test (Extended Data Fig. 1). 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.

predicts large areas of high performance. During adaptation, these areas disappear because the behaviors do not work well on the damaged robot. Intelligent Trial and Error quickly identifies one of the few, remaining, high-performance behaviors (Fig. 4 and Extended Data Fig. 5 and 6).

The same damage recovery approach can be applied to any robot, such as a robotic arm. We tested 14 different damage conditions with a planar, 8-joint robotic arm (Fig. 3D-F and Extended Data Fig. 7). The behavior-performance map’s behavioral dimensions are the  $x, y$  position of the end-effector and the performance measure is minimizing the variance of the 8 specified motor angles (Supplementary Methods). During adaptation, performance is measured as distance to the target. Like with the hexapod robot, our approach discovers a compensatory behavior in less than 2 minutes, usually in less than 30 seconds, and with fewer than 10 trials (Fig. 3F and Extended Data Fig. 7).

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 every possible damage condition: 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<sup>24</sup>, 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<sup>23</sup>, and there is strong evidence that animal brains learn probability distributions, combine them with prior knowledge, and act as Bayesian optimizers<sup>25,26</sup>.

An additional parallel is that Intelligent Trial and Error primes the robot for creativity during a motionless period, after which

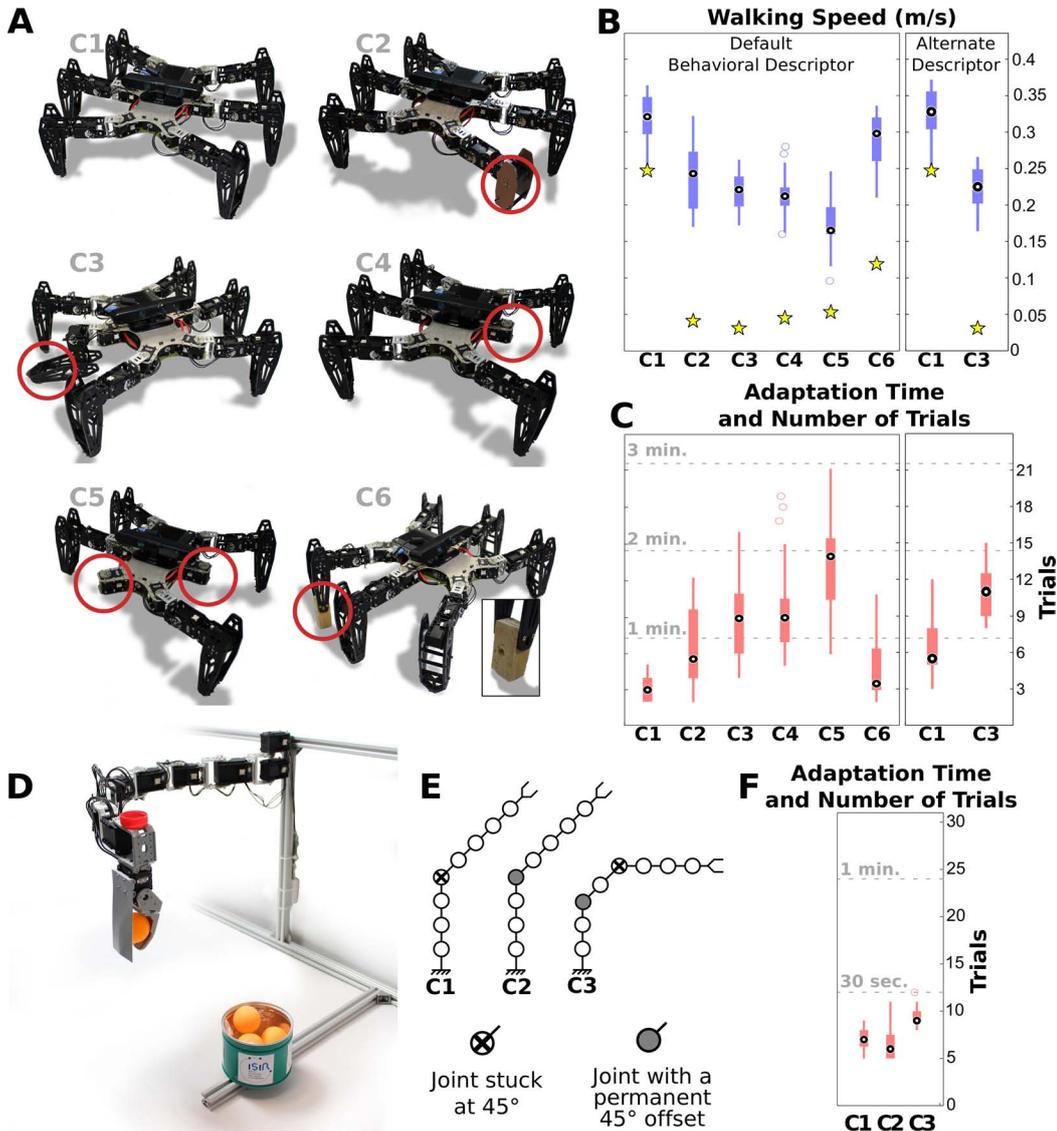
the generated ideas are tested. This process is reminiscent of the finding that some animals start the day with new ideas that they may quickly disregard after experimenting with them<sup>27</sup>, and more generally, that sleep improves creativity on cognitive tasks<sup>28</sup>. A final parallel is 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<sup>29,12</sup>. All told, we have 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.

**Supplementary Information and methods** are appended at the end of this document.

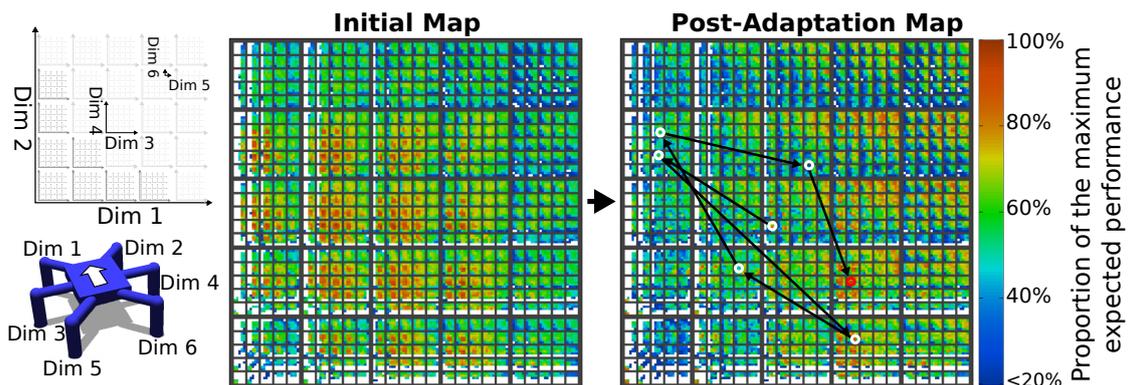
**Acknowledgments.** Thanks to Luigi Tedesco, Stéphane Doncieux, Nicolas Bredeche, Shimon Whiteson, Roberto Calandra, Jacques Droulez, Pierre Bessière, Florian Lesaint, Charles Thurat, Serena Ivaldi, Jingyu Li, Joost Huizinga, Roby Velez, Henk Mengistu, Tim Clune, and Anh Nguyen for helpful feedback 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), the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement number 637972), and a Direction Générale de l’Armement (DGA) scholarship to A.C.

**Author Contributions** A.C. and J.-B. M designed the study. A.C. and D.T. performed the experiments. A.C., J.-B. M, D.T. and J.C. discussed additional experiments, analyzed the results, and wrote the paper.



**Figure 3 | (A) Conditions tested on the physical hexapod 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. The central mark is the median, the edges of the box are the 25<sup>th</sup> and 75<sup>th</sup> percentiles, the whiskers extend to the most extreme data points not considered outliers, and outliers are plotted individually. Yellow stars represent the performance of the handmade reference tripod gait (Supplementary Methods). Conditions C1-C6 are tested 5 times each for 8 independently created behavior-performance maps with the “duty factor” behavior description (i.e. 40 experiments per damage condition, Supplementary Methods). Damage conditions C1 and C3 are also tested 5 times each for 8 independently created behavior-performance maps with the “body orientation” behavior description (Supplementary Methods). **(C) Time and number of trials required to adapt.** Box plots represent Intelligent Trial and Error. **(D) Robotic arm experiment.** The 8-joint, planar robot has to drop a ball into a bin. **(E) Example conditions tested on the physical robotic arm.** C1: One joint is stuck at 45 degrees. C2: One joint has a permanent 45-degree offset. C3: One broken and one offset joint. A total of 14 conditions were tested (Extended Data Fig. 7). **(F) Time and number of trials required to reach within 5 cm of the bin center.** Each condition is tested with 15 independently created behavior-performance maps.



**Figure 4 | An example behavior-performance map.** This map 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 map creation at that point in the behavior space. The matrices visualize the six-dimensional behavioral space in two dimensions according to the legend in the top-left. The behavior-performance map is created with a simulated robot (bottom left) in the Open Dynamics Engine physics simulator (<http://www.ode.org>). The left matrix is a pre-adaptation map produced by the map creation algorithm. During adaptation, the map 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 map after a compensatory behavior is discovered. The arrows and 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 for the damaged robot 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.

**Author information** Correspondence and requests for materials should be addressed to J.-B. M. (email: jean-baptiste.mouret@inria.fr).

1. J. G. Bellingham and K. Rajan. "Robotics in remote and hostile environments." In: *Science* 318.5853 (Nov. 2007), pp. 1098–102. ISSN: 1095-9203. DOI: 10.1126/science.1146230.
2. D. R. Yoerger. "Underwater robotics". In: *Springer handbook of robotics*. Springer, 2008, pp. 987–1008.
3. E. Broadbent, R. Stafford, and B. MacDonald. "Acceptance of healthcare robots for the older population: review and future directions". In: *International Journal of Social Robotics* 1.4 (2009), pp. 319–330.
4. K. Sanderson. "Mars rover Spirit (2003-10)". In: *Nature* 463.7281 (2010), p. 600.
5. J. Carlson and R. R. Murphy. "How UGVs physically fail in the field". In: *IEEE Transactions on Robotics* 21.3 (2005), pp. 423–437.
6. M. Blanke and J. Schröder. *Diagnosis and fault-tolerant control*. Springer, 2006.
7. B. Siciliano and O. Khatib. *Springer handbook of robotics*. Springer, 2008.
8. R. R. Murphy. "Trial by fire [rescue robots]". In: *Robotics & Automation Magazine, IEEE* 11.3 (2004), pp. 50–61.
9. K. Nagatani, S. Kiribayashi, Y. Okada, K. Otake, K. Yoshida, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, M. Fukushima, and S. Kawatsuma. "Emergency response to the nuclear accident at the Fukushima Daiichi Nuclear Power Plants using mobile rescue robots". In: *Journal of Field Robotics* 30.1 (2013), pp. 44–63.
10. S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, et al. "Stanley: The robot that won the DARPA Grand Challenge". In: *Journal of field Robotics* 23.9 (2006), pp. 661–692.
11. V. Verma, G. Gordon, R. Simmons, and S. Thrun. "Real-time fault diagnosis". In: *Robotics & Automation Magazine* 11.2 (2004), pp. 56–66.
12. J. Bongard, V. Zykov, and H. Lipson. "Resilient machines through continuous self-modeling". In: *Science* 314.5802 (2006), pp. 1118–1121.
13. W. G. Fenton, T. M. McGinnity, and L. P. Maguire. "Fault diagnosis of electronic systems using intelligent techniques: a review". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 31.3 (2001), pp. 269–281.
14. J. Kluger and J. Lovell. *Apollo 13*. Mariner Books, 2006. ISBN: 978-0618619580.
15. S. L. Jarvis, D. R. Worley, S. M. Hogy, A. E. Hill, K. K. Haussler, and R. F. Reiser II. "Kinematic and kinetic analysis of dogs during trotting after amputation of a thoracic limb". In: *American journal of veterinary research* 74.9 (2013), pp. 1155–1163.
16. A. Fuchs, B. Goldner, I. Nolte, and N. Schilling. "Ground reaction force adaptations to tripedal locomotion in dogs." In: *Veterinary journal* 201.3 (Sept. 2014), pp. 307–15. ISSN: 1532-2971. DOI: 10.1016/j.tvjl.2014.05.012.
17. J. Kober, J. A. Bagnell, and J. Peters. "Reinforcement learning in robotics: a survey". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274. DOI: 10.1177/0278364913495721.
18. B. D. Argall, S. Chernova, M. Veloso, and B. Browning. "A survey of robot learning from demonstration". In: *Robotics and autonomous systems* 57.5 (2009), pp. 469–483.
19. E. Thelen. "Motor development: a new synthesis". In: *American psychologist* (1995), pp. 79–95.
20. M. Santello. "Postural hand synergies for tool use". In: *The Journal of Neuroscience* 18.23 (1998), pp. 10105–10115.
21. C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. MIT Press, 2006. ISBN: 0-262-18253-X.
22. J. Mockus. *Bayesian approach to global optimization: theory and applications*. Kluwer Academic, 2013.
23. A. Borji and L. Itti. "Bayesian optimization explains human active search". In: *Advances in Neural Information Processing Systems 26 (NIPS)*. 2013, pp. 55–63.
24. S. Benson-Amram and K. E. Holekamp. "Innovative problem solving by wild spotted hyenas". In: *Proceedings of the Royal Society B: Biological Sciences* 279.1744 (2012), pp. 4087–4095.
25. A. Pouget, J. M. Beck, W. J. Ma, and P. E. Latham. "Probabilistic brains: knowns and unknowns". In: *Nature neuroscience* 16.9 (2013), pp. 1170–1178.
26. K. P. Körding and D. M. Wolpert. "Bayesian integration in sensorimotor learning". In: *Nature* 427.6971 (2004), pp. 244–247.
27. S. Derégnaucourt, P. P. Mitra, O. Fehér, C. Pytte, and O. Tchernichovski. "How sleep affects the developmental learning of bird song". In: *Nature* 433.7027 (2005), pp. 710–716.
28. U. Wagner, S. Gais, H. Haider, R. Verleger, and J. Born. "Sleep inspires insight". In: *Nature* 427.6972 (2004), pp. 352–355.
29. M. Ito. "Control of mental activities by internal models in the cerebellum". In: *Nature Reviews Neuroscience* 9.4 (2008), pp. 304–313.

# 1 Methods

## Notations

- $\mathbf{c}$ : Parameters of a controller (vector)
- $\mathbf{x}$ : A location in a discrete behavioral space (i.e. a type of behavior) (vector)
- $\chi$ : A location in a discrete behavioral space that has been tested on the physical robot (vector)
- $\mathcal{P}$ : Behavior-performance map (stores performance) (associative table)
- $\mathcal{C}$ : Behavior-performance map (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)
- $\chi_{1:t}$ : All previously tested behavioral descriptors at time  $t$  (vector of vectors)
- $\mathbf{P}_{1:t}$ : Performance in reality of all the candidate solutions tested on the robot up to time  $t$  (vector)
- $\mathcal{P}(\chi_{1:t})$ : Performance in the behavior-performance map for all the candidate solutions tested on the robot up to time  $t$  (vector)
- $f()$ : Performance function (unknown by the algorithm) (function)
- $\sigma_{noise}^2$ : Observation noise (a user-specified parameter) (scalar)
- $k(\mathbf{x}, \mathbf{x})$ : Kernel function (see section “kernel function”) (function)
- $\mathbf{K}$ : Kernel matrix (matrix)
- $\mathbf{k}$ : Kernel vector  $[k(\mathbf{x}, \chi_1), k(\mathbf{x}, \chi_2), \dots, k(\mathbf{x}, \chi_t)]$  (vector)
- $\mu_t(\mathbf{x})$ : Predicted performance for  $\mathbf{x}$  (i.e. the mean of the Gaussian process) (function)
- $\sigma_t^2(\mathbf{x})$ : Standard deviation for  $\mathbf{x}$  in the Gaussian process (function)

### 1.1 Intelligent Trial and Error algorithm (IT&E)

The Intelligent Trial and Error Algorithm consists of two major steps (Extended Data Fig. 1): the behavior-performance map creation step and the adaptation step (while here we focus on damage recovery, Intelligent Trial and Error can search for any type of required adaptation, such as learning an initial gait for an undamaged robot, adapting to new environments, etc.). The behavior-performance map 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 the next section. The adaptation step is accomplished via a second new algorithm introduced in this paper called the map-based Bayesian optimization algorithm (M-BOA), which is explained in the “Adaptation Step” section below.

### 1.2 Behavior-performance map creation (via the MAP-Elites algorithm)

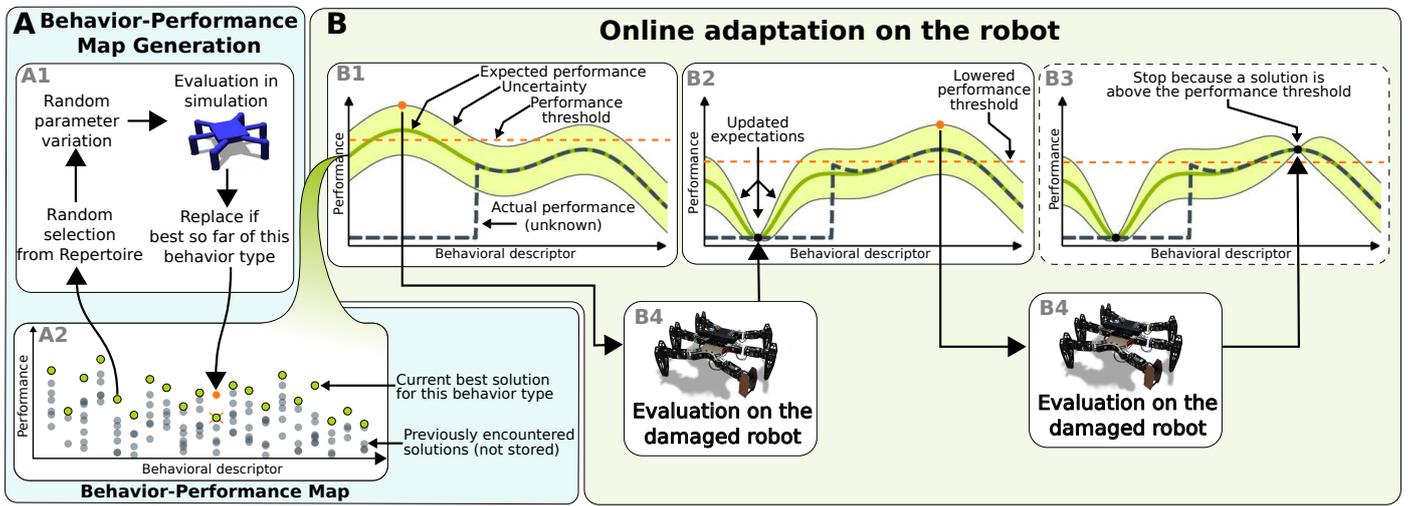
The behavior-performance map is created by a new algorithm we introduce in this paper called the multi-dimensional archive of phenotypic elites (MAP-Elites) algorithm. MAP-Elites searches for the highest-performing solution for each point in a user-defined space: the user chooses the dimensions of the 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 behavior-performance 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 behavior-performance map 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 randomly altering a high-performing solution of one type can produce a high-performing solution of a different type (see Extended Data Fig. 10 and Supplementary Experiment S4). 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 at that location in the behavior-performance map, then it is added to the behavior-performance map, 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 behavior-performance map at that location, then the newly generated candidate solution is added at that location.

Once this initialization step is finished, Map-Elites enters a loop that is similar to stochastic, population-based, optimization algorithms, such as evolutionary algorithms<sup>30</sup>: the solutions that are in the behavior-performance map 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 40 million iterations. Because MAP-Elites is a stochastic



**Extended Data Figure 1 | An overview of the Intelligent Trial and Error Algorithm. (A) Behavior-performance map creation.** After being initialized with random controllers, the behavioral map (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 map (here, after 40 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 map involved 40 million iterations of (A1), which lasted roughly two weeks on one multi-core computer (Supplementary Methods, section “Running time”). **(B) Adaptation.** (B1) Each behavior from the behavior-performance map 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 (Methods, section “acquisition function”). 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, see Methods, section “kernel function”), the results of which can be seen in (B2). The process is then repeated until performance on the damaged robot is 90% or greater of the maximum expected performance for any behavior (B3). This performance threshold (orange dashed line) lowers as the maximum 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).

search process, each resultant behavior-performance 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.

The pseudo-code of the algorithm is available in Supplementary Figure 8. More details and experiments about MAP-Elites are available in (Mouret and Clune, 2015)<sup>31</sup>.

### 1.3 Adaptation step (via M-BOA: the map-based Bayesian optimization algorithm)

The adaptation step is accomplished via a Bayesian optimization algorithm seeded with a behavior-performance map. We call this approach a map-based Bayesian optimization algorithm, or M-BOA.

Bayesian optimization is a model-based, black-box optimization algorithm that is tailored for very expensive objective functions (a.k.a. cost functions)<sup>32,33,22,34,35,23</sup>. 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<sup>36,37,38</sup>, kriging<sup>39</sup>, or DACE<sup>40,41</sup>), 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<sup>22</sup>, 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<sup>21</sup>, which is a common choice for Bayesian optimization<sup>42,35,33,32</sup>. Gaussian processes are particularly interesting for regression because they not only model the cost function, but also the uncer-

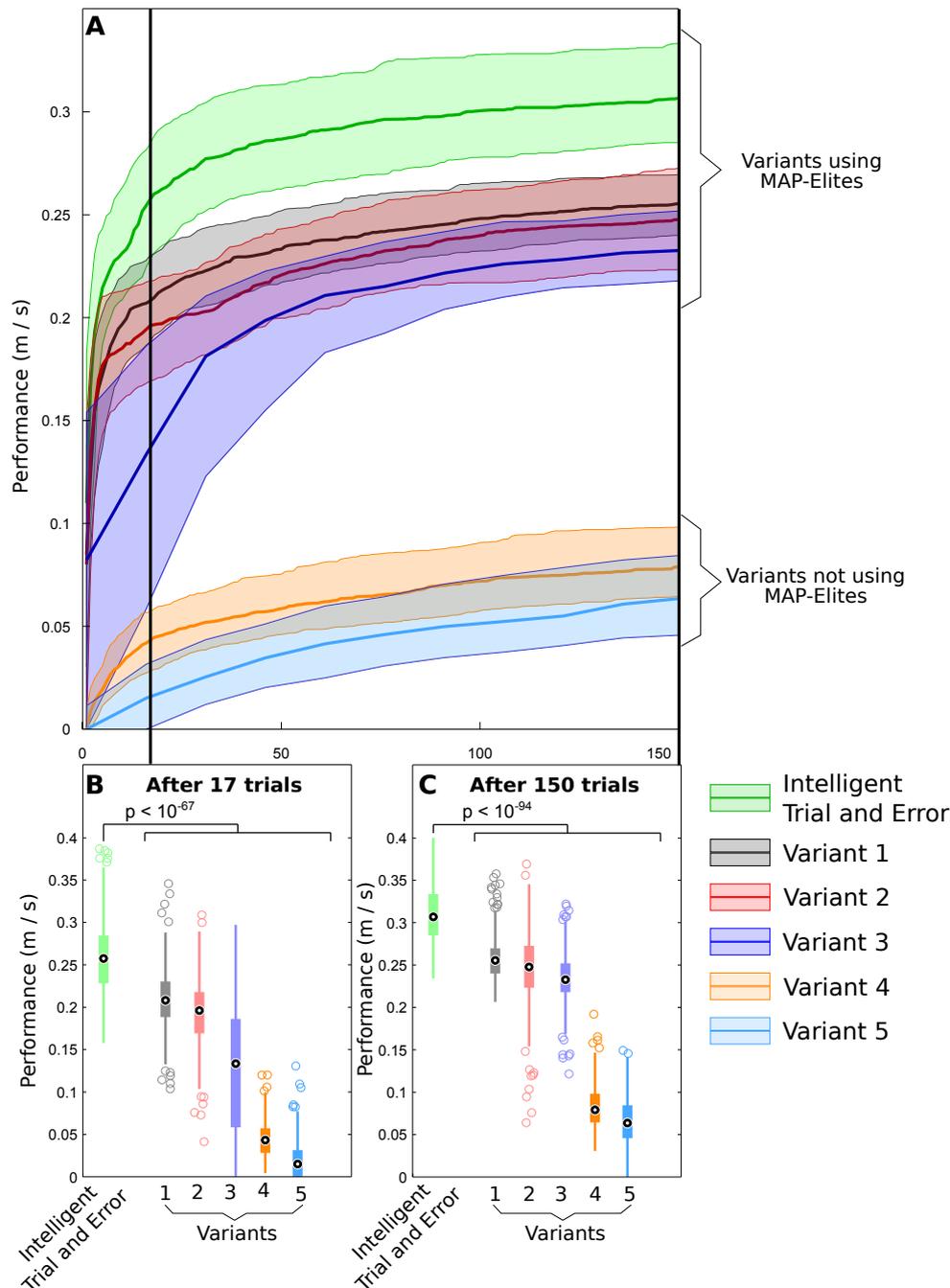
tainty associated with each prediction. For a cost function  $f$ , usually unknown, a Gaussian process defines the probability distribution of the possible values  $f(\mathbf{x})$  for each point  $\mathbf{x}$ . These probability distributions are Gaussian, and are therefore defined by a mean ( $\mu$ ) and a standard deviation ( $\sigma$ ). However,  $\mu$  and  $\sigma$  can be different for each  $\mathbf{x}$ ; we therefore define a probability distribution *over functions*:

$$P(f(\mathbf{x})|\mathbf{x}) = \mathcal{N}(\mu(\mathbf{x}), \sigma^2(\mathbf{x})) \quad (1)$$

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

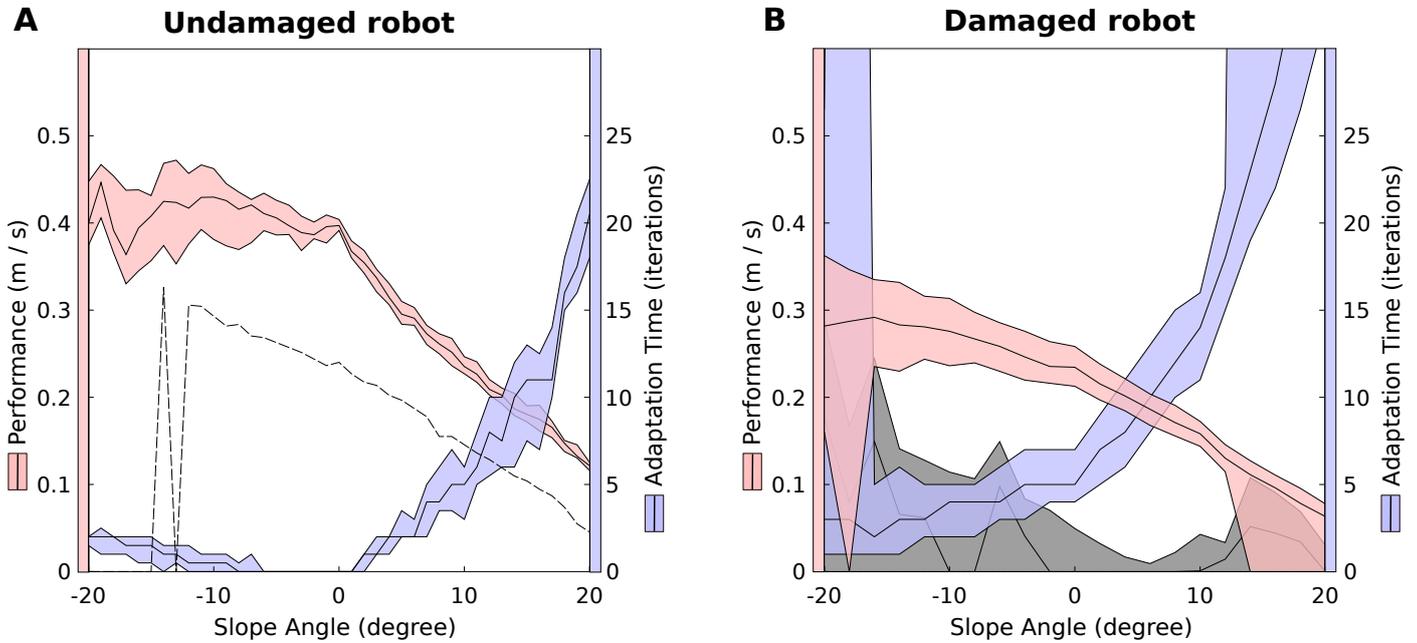
To estimate  $\mu(\mathbf{x})$  and  $\sigma(\mathbf{x})$ , we need to fit the Gaussian process to the data. To do so, we assume that each observation  $f(\chi)$  is a sample from a normal distribution. If we have a data set made of several observations, that is,  $f(\chi_1), f(\chi_2), \dots, f(\chi_t)$ , then the vector  $[f(\chi_1), f(\chi_2), \dots, f(\chi_t)]$  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  $\chi_1$  and  $\chi_2$  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  $\chi_1$  and  $\chi_2$  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  $\chi_1$  and  $\chi_2$  (see the “kernel function” sub-section below).

Given a set of observations  $\mathbf{P}_{1:t} = f(\chi_{1:t})$  and a sampling noise  $\sigma_{noise}^2$  (which is a user-specified parameter), the Gaussian



Variant	Behavior-performance map 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)
Variant 5	none	none	policy gradient	Kohl et al. (2004)

**Extended Data Figure 2 | The contribution of each subcomponent of the Intelligent Trial and Error Algorithm. (A) Adaptation progress versus the number of robot trials.** The walking speed achieved with Intelligent Trial and Error and several “knockout” variants that are missing one of the algorithm’s key components. Some variants (4 and 5) correspond to state-of-the-art learning algorithms (policy gradient: Kohl et al. 2004; Bayesian optimization: Lizotte et al. 2007, Tesch et al., 2011, Calandra et al. 2014.). The bold lines represent the medians and the colored areas extend to the 25<sup>th</sup> and 75<sup>th</sup> percentiles. **(B, C) Adaptation performance after 17 and 150 trials.** Shown is the the speed of the compensatory behavior discovered by each algorithm after 17 and 150 evaluations on the robot, respectively. For all panels, data are pooled across six damage conditions (the removal of each of the 6 legs in turn). See Supplementary Experiment S2 for methods and analysis.



**Extended Data Figure 3 | The Intelligent Trial and Error Algorithm is robust to environmental changes.** Each plot shows both the performance and required adaptation time for Intelligent Trial and Error when the robot must adapt to walk on terrains of different slopes. **(A) Adaptation performance on 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) Adaptation performance on a damaged robot.** The robot is damaged by having each of the six legs removed in six different damage scenarios. Data are pooled from all six of these damage conditions. 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 25<sup>th</sup> and 75<sup>th</sup> 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 25<sup>th</sup> and 75<sup>th</sup> percentiles (black colored area) are shown. See Supplementary Experiment S3 for methods and analysis.

process is computed as follows<sup>33,21</sup>:

$$P(f(\mathbf{x})|\mathbf{P}_{1:t}, \mathbf{x}) = \mathcal{N}(\mu_t(\mathbf{x}), \sigma_t^2(\mathbf{x}))$$

where :

$$\mu_t(\mathbf{x}) = \mathbf{k}^T \mathbf{K}^{-1} \mathbf{P}_{1:t}$$

$$\sigma_t^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}^T \mathbf{K}^{-1} \mathbf{k}$$

$$\mathbf{K} = \begin{bmatrix} k(\chi_1, \chi_1) & \cdots & k(\chi_1, \chi_t) \\ \vdots & \ddots & \vdots \\ k(\chi_t, \chi_1) & \cdots & k(\chi_t, \chi_t) \end{bmatrix} + \sigma_{noise}^2 \mathbf{I}$$

$$\mathbf{k} = [ k(\mathbf{x}, \chi_1) \quad k(\mathbf{x}, \chi_2) \quad \cdots \quad k(\mathbf{x}, \chi_t) ]$$

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  $\chi$  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 map-based Bayesian optimization algorithm (M-BOA) 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, M-BOA models the *difference* between the prediction of the behavior-performance map and the actual performance on the real robot, instead of directly modeling the objective function. This idea is incorporated into the Gaussian process by modifying

the update equation for the mean function ( $\mu_t(\mathbf{x})$ , equation 2):

$$\mu_t(\mathbf{x}) = \mathcal{P}(\mathbf{x}) + \mathbf{k}^T \mathbf{K}^{-1} (\mathbf{P}_{1:t} - \mathcal{P}(\chi_{1:t})) \quad (3)$$

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. 2) by  $\mathbf{P}_{1:t} - \mathcal{P}(\chi_{1:t})$  (eq. 3) means that the Gaussian process models the difference between the actual performance  $\mathbf{P}_{1:t}$  and the performance from the behavior-performance map  $\mathcal{P}(\chi_{1:t})$ . The term  $\mathcal{P}(\mathbf{x})$  is the prediction of the behavior-performance map. M-BOA therefore starts with the prediction from the behavior-performance map and corrects it with the Gaussian process.

The pseudo-code of the algorithm is available in Supplementary Figure 8.

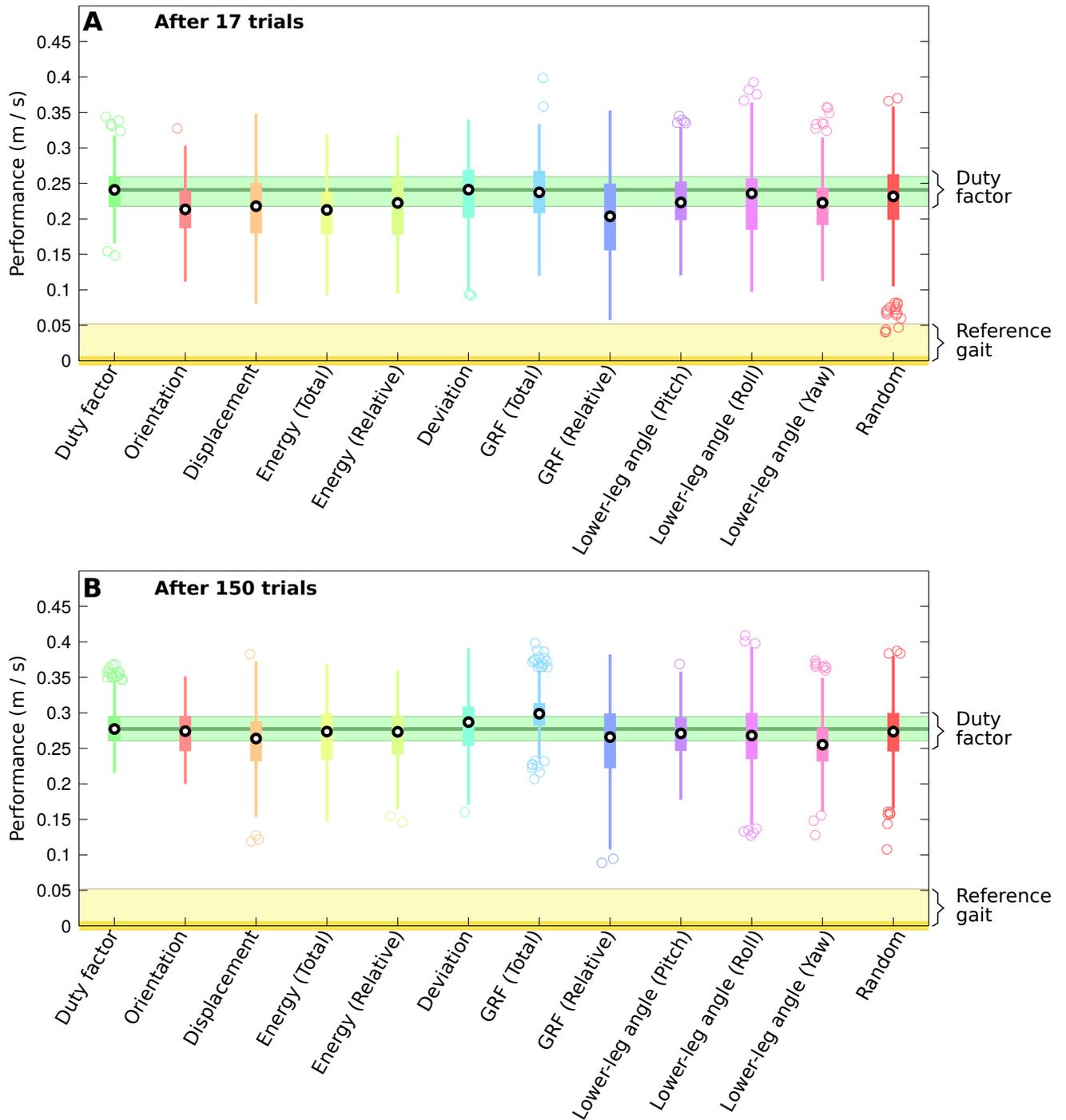
**Kernel function** The kernel function is the covariance function of the Gaussian process. It defines the influence of a controller’s performance (on the physical robot) on the performance and confidence estimations of not-yet-tested controllers in the behavior-performance map that are nearby in behavior space to the tested controller (Extended Data Fig. 9a).

The Squared Exponential covariance function and the Matérn kernel are the most common kernels for Gaussian processes<sup>33,34,21</sup>. 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  $\rho$ , Extended Data Fig. 9a), but also the rate at which distance effects decrease (as a function of parameter  $\nu$ ).

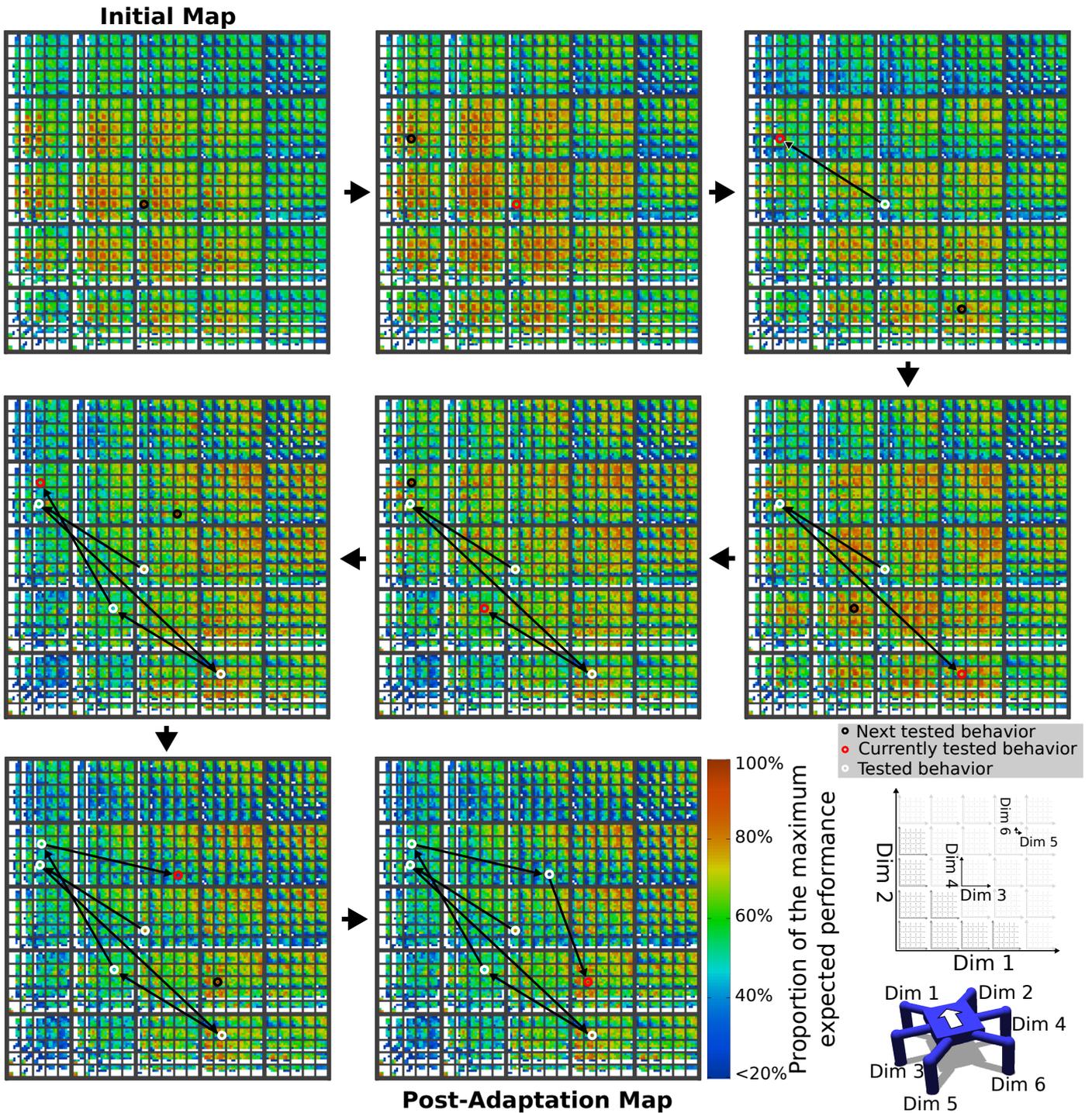
The Matérn kernel function is computed as follows<sup>43,44</sup> (with  $\nu = 5/2$ ):

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

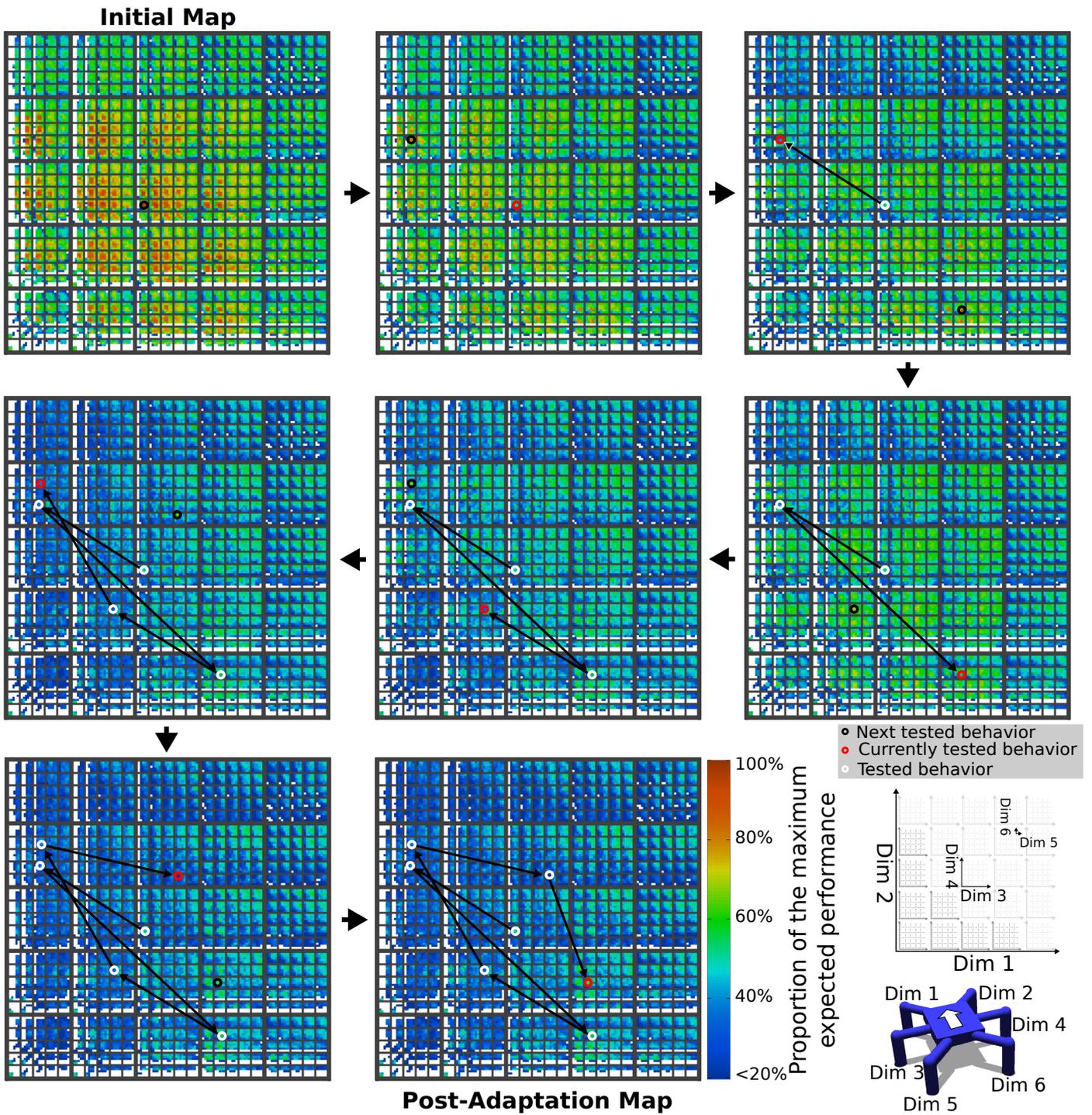
where  $d(\mathbf{x}_1, \mathbf{x}_2)$  is the Euclidean distance in behavior space. (4)



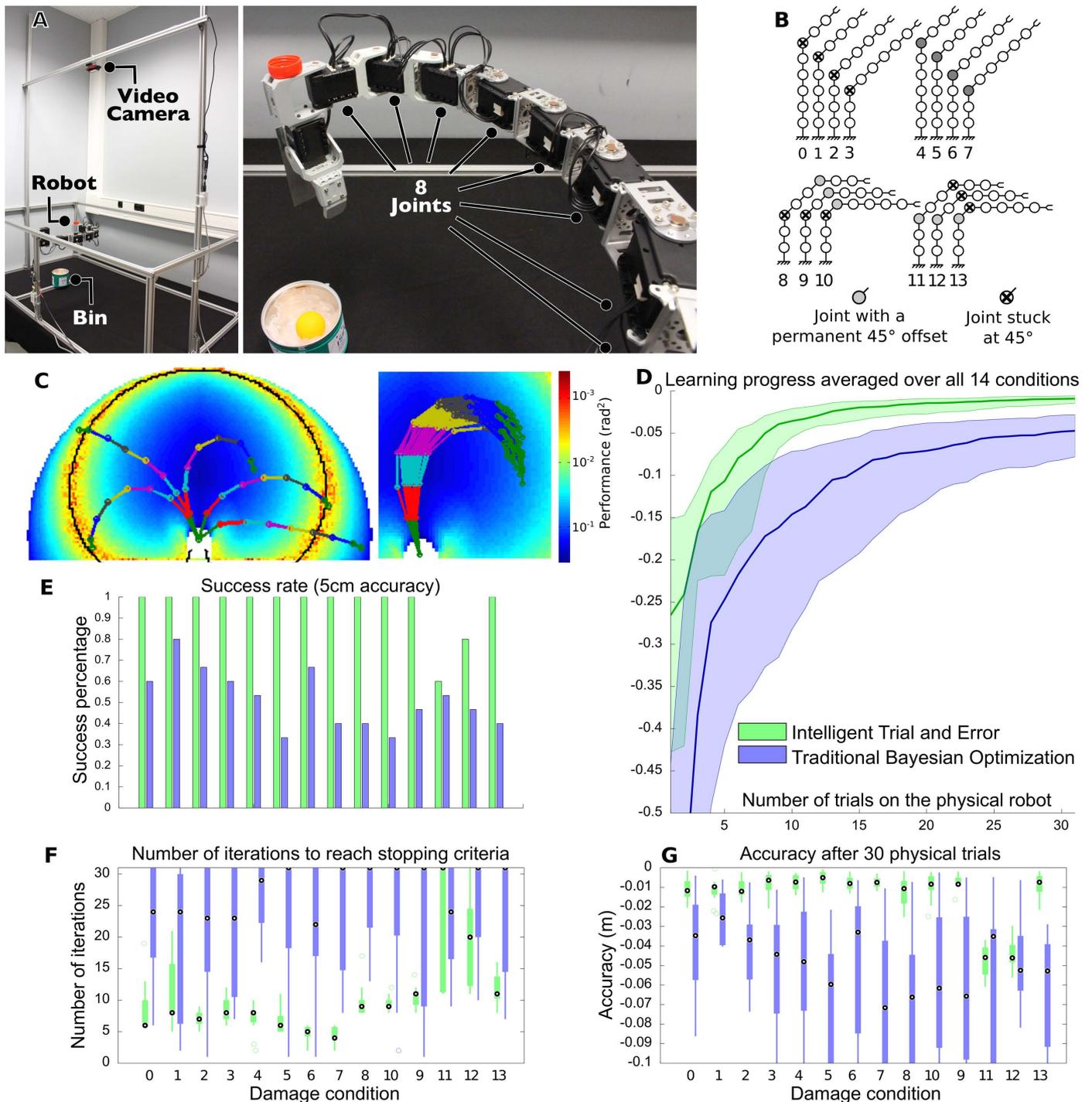
**Extended Data Figure 4 | The Intelligent Trial and Error Algorithm is largely robust to alternate choices of behavior descriptors.** (A, B) The speed of the compensatory behavior discovered by Intelligent Trial and Error for various choices of behavior descriptors. Performance is plotted after 17 and 150 evaluations in panels A and B, respectively. Experiments were performed on a simulated, damaged hexapod. The damaged robot has each of its six legs removed in six different damage scenarios. Data are pooled across all six damage conditions. As described in Supplementary Experiment S5, the evaluated behavior descriptors characterize the following: (i) Time each leg is in contact with the ground (**Duty factor**); (ii) Orientation of the robot frame (**Orientation**); (iii) Instantaneous velocity of the robot (**Displacement**); (iv) Energy expended by the robot in walking (**Energy (Total)**, **Energy (Relative)**); (v) Deviation from a straight line (**Deviation**); (vi) Ground reaction force on each leg (**GRF (Total)**, **GRF (Relative)**); (vii) The angle of each leg when it touches the ground (**Lower-leg angle (Pitch)**, **Lower-leg angle (Roll)**, **Lower-leg angle (Yaw)**); and (viii) A random selection without replacement from subcomponents of all the available behavior descriptors (i-vii) (**Random**). For the hand-designed reference gait (yellow) and the compensatory gaits found by the default duty factor behavior descriptor (green), the bold lines represent the medians and the colored areas extend to the 25<sup>th</sup> and 75<sup>th</sup> percentiles of the data. For the other treatments, including the duty factor treatment, black circles represent the median, the colored area extends to the 25<sup>th</sup> and 75<sup>th</sup> percentiles of the data, and the colored circles are outliers. See Supplementary Experiment S5 for methods and analysis.



Extended Data Figure 5 | How the behavior performance map is explored to discover a compensatory behavior (normalized each iteration to highlight the *range* of remaining performance predictions). Colors represent the performance prediction for each point in the map relative to the highest performing prediction in the map at that step of the process. A black circle indicates the next behavior to be tested on the physical robot. A red circle indicates the behavior that was just tested (note that the performance predictions surrounding it have changed versus the previous panel). Arrows reveal the order that points have been explored. The red circle in the last map is the final, selected, compensatory behavior. In this scenario, the robot loses leg number 3. The six dimensional space is visualized according to the inset legend.



Extended Data Figure 6 | How the behavior performance map is explored to discover a compensatory behavior (non-normalized to highlight that performance predictions decrease as it is discovered that predictions from the simulated, undamaged robot do not work well on the damaged robot). Colors represent the performance prediction for each point in the map relative to the highest performing prediction in the *first* map. A black circle indicates the next behavior to be tested on the physical robot. A red circle indicates the behavior that was just tested (note that the performance predictions surrounding it have changed versus the previous panel). Arrows reveal the order that points have been explored. The red circle in the last map in the sequence is the final, selected, compensatory behavior. In this scenario, the robot loses leg number 3. The six dimensional space is visualized according to the inset legend. The data visualized in this figure are identical to those in the previous figure: the difference is simply whether the data are renormalized for each new map in the sequence.



**Extended Data Figure 7 | Intelligent Trial and Error works on a completely different type of robot: supplementary data from the robotic arm experiment.** (A) The robotic arm experimental setup. (B) Tested damage conditions. (C) Example of behavior performance maps (colormaps) and behaviors (overlaid arm configurations) obtained with MAP-Elites. Left: A typical behavior-performance map produced by MAP-Elites with 5 example behaviors, where a behavior is described by the angle of each of the 8 joints. The color of each point is a function of its performance, which is defined as having low variance in the joint angles (i.e. a zigzag arm is lower performing than a straighter arm that reaches the same point). Right: Neighboring points in the map tend to have similar behaviors, thanks to the performance function, which would penalize more jagged ways of reaching those points. That neighbors have similar behaviors justifies updating predictions about the performance of nearby behaviors after a testing a single behavior on the real (damaged) robot. (D) Performance vs. trial number for Intelligent Trial and Error and traditional Bayesian optimization. The experiment was conducted on the physical robot, with 15 independent replications for each of the 14 damage conditions. Performance is pooled from all of these  $14 \times 15 = 210$  experiments. (E) Success for each damage condition. Shown is the success rate for the 15 replications for each damage condition, defined as the percentage of replicates in which the robot reaches within 5 cm of the bin center. (F) Trials required to adapt. Shown is the number of iterations required to reach within 5 cm of the basket center. (G) Accuracy after 30 physical trials. Performance after 30 physical trials for each damage condition (with the stopping criterion disabled). See Supplementary Experiment S1 for methods and analysis.

**procedure INTELLIGENT TRIAL AND ERROR ALGORITHM (IT&E)**

Before the mission:

CREATE BEHAVIOR-PERFORMANCE MAP (VIA THE MAP-ELITES ALGORITHM IN SIMULATION)

**while** In mission **do**

**if** Significant performance fall **then**

ADAPTATION STEP (VIA M-BOA ALGORITHM)

**procedure MAP-ELITES ALGORITHM**

$(\mathcal{P} \leftarrow \emptyset, \mathcal{C} \leftarrow \emptyset)$

▷ Creation of an empty behavior-performance map (empty N-dimensional grid).

**for** iter = 1 → I **do**

▷ Repeat during I iterations (here we choose I = 40 million iterations).

**if** iter < 400 **then**

$\mathbf{c}' \leftarrow \text{random\_controller}()$

▷ The first 400 controllers are generated randomly.

**else**

▷ The next controllers are generated using the map.

$\mathbf{c} \leftarrow \text{random\_selection}(\mathcal{C})$

▷ Randomly select a controller c in the map.

$\mathbf{c}' \leftarrow \text{random\_variation}(\mathbf{c})$

▷ Create a randomly modified copy of c.

$\mathbf{x}' \leftarrow \text{behavioral\_descriptor}(\text{simu}(\mathbf{c}'))$

▷ Simulate the controller and record its behavioral descriptor.

$p' \leftarrow \text{performance}(\text{simu}(\mathbf{c}'))$

▷ Record its performance.

**if**  $\mathcal{P}(\mathbf{x}') = \emptyset$  or  $\mathcal{P}(\mathbf{x}') < p'$  **then**

▷ If the cell is empty or if p' is better than the current stored performance.

$\mathcal{P}(\mathbf{x}') \leftarrow p'$

▷ Store the performance of c' in the behavior-performance map according

▷ to its behavioral descriptor x'.

$\mathcal{C}(\mathbf{x}') \leftarrow \mathbf{c}'$

▷ Associate the controller with its behavioral descriptor.

**return** behavior-performance map ( $\mathcal{P}$  and  $\mathcal{C}$ )

**procedure M-BOA (MAP-BASED BAYESIAN OPTIMIZATION ALGORITHM)**

$\forall \mathbf{x} \in \text{map}:$

▷ 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 the map.

$\sigma_0^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x})$

▷ 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).

$P_{t+1} \leftarrow \text{performance}(\text{physical\_robot}(\mathcal{C}(\chi_{t+1})))$ .

▷ 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

$\mu_{t+1}(\mathbf{x}) = \mathcal{P}(\mathbf{x}) + \mathbf{k}^\top \mathbf{K}^{-1} (\mathbf{P}_{1:t+1} - \mathcal{P}(\chi_{1:t+1}))$

▷ Update the mean.

$\sigma_{t+1}^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}^\top \mathbf{K}^{-1} \mathbf{k}$

▷ Update the variance.

$$\mathbf{K} = \begin{bmatrix} k(\chi_1, \chi_1) & \cdots & k(\chi_1, \chi_{t+1}) \\ \vdots & \ddots & \vdots \\ k(\chi_{t+1}, \chi_1) & \cdots & k(\chi_{t+1}, \chi_{t+1}) \end{bmatrix} + \sigma_{noise}^2 \mathbf{I}$$

▷ Compute the observations' correlation matrix.

$$\mathbf{k} = [ k(\mathbf{x}, \chi_1) \quad k(\mathbf{x}, \chi_2) \quad \cdots \quad k(\mathbf{x}, \chi_{t+1}) ]$$

▷ Compute the  $\mathbf{x}$  vs. observation correlation vector.

Extended Data Figure 8 | Pseudo-code for the Intelligent Trial and Error Algorithm, the MAP-Elites algorithm, and the Map-based Bayesian Optimization Algorithm (M-BOA). Notations are described at the beginning of the methods section.

Because the model update step directly depends on  $\rho$ , it is one of the most critical parameters of the Intelligent Trial and Error Algorithm. We selected its value after extensive experiments in simulation (Extended Data Fig. 9 and section 1.6).

**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<sup>45</sup>. For the specific behavior space in the example problem in this paper, though, the discretized search space of the behavior-performance map is small enough that we can exhaustively compute the acquisition value of each solution of the behavior-performance map 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)<sup>33,42</sup>. We chose UCB because it provided the best results in several previous studies<sup>33,42</sup>. The equation for UCB is:

$$\mathbf{x}_{t+1} = \arg \max_{\mathbf{x}} (\mu_t(\mathbf{x}) + \kappa \sigma_t(\mathbf{x})) \quad (5)$$

where  $\kappa$  is a user-defined parameter that tunes the tradeoff between exploration and exploitation.

The acquisition function handles the exploitation/exploration trade-off of the adaptation (M-BOA) step. In the UCB function (Eq. 5), 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  $\kappa$  factor. With a low  $\kappa$ , the algorithm will choose solutions that are expected to be high-performing. Conversely, with a high  $\kappa$ , the algorithm will focus its search on unexplored areas of the search space that may have high-performing solutions. The  $\kappa$  factor enables fine adjustments to the exploitation/exploration trade-off of the M-BOA algorithm (the adaptation step). We describe how we chose the  $\kappa$  value in supplementary methods, section 1.6.

**Code availability** 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](http://pages.isir.upmc.fr/~mouret/code/ite_source_code.tar.gz) An implementation of the Bayesian optimization algorithm is freely available on: <http://github.com/jbmouret/limbo>

## 1.4 Hexapod Experiment

**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<sup>146</sup> from the robot operating system (ROS) framework<sup>247</sup>.

**Simulator** The simulator is a dynamic physics simulation of the undamaged 6-legged robot on flat ground (Fig. 4). We weighted each segment of the leg and the body of the real robot, and we

<sup>1</sup>[http://wiki.ros.org/ccny\\_openni\\_launch](http://wiki.ros.org/ccny_openni_launch)

<sup>2</sup><http://www.ros.org>

used the same masses for the simulations. The simulator is based on the Open Dynamics Engine (ODE, <http://www.ode.org>).

**Parametrized controller** The angular position of each DOF is governed by a periodic function  $\gamma$  parametrized by its amplitude  $\alpha$ , its phase  $\phi$ , and its duty cycle  $\tau$  (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  $\alpha$ , and duty cycle  $\tau$ . This signal is then smoothed via a Gaussian filter in order to remove sharp transitions, and is then shifted according to the phase  $\phi$ .

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, angles 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<sup>48</sup> and evolved neural networks<sup>49,50,51,52</sup>.

**Reference controller** Our reference controller is a classic tripod gait<sup>7,53,54,55,56,57</sup>. It involves two tripods: legs 1-4-5 and legs 2-3-6 (Fig. 4). This controller 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 tripod. This gait is static, fast, and similar to insect gaits<sup>53,58</sup>.

Table S1 shows the 36 parameters of the reference controller. The amplitude orientation parameters ( $\alpha_{i_1}$ ) are set to 1 to produce the fastest possible gait, while the amplitude elevation parameters ( $\alpha_{i_2}$ ) are set to a small value (0.25) to keep the gait stable. The phase elevation parameters ( $\phi_{i_2}$ ) define two tripods: 0.25 for legs 2-3-5; 0.75 for legs 1-4-5. To achieve a cyclic motion of the leg, the phase orientation values ( $\phi_{i_1}$ ) are chosen by subtracting 0.25 to the phase elevation values ( $\phi_{i_2}$ ), plus a 0.5 shift for legs 1-3-5, which are on the left side of the robot. All the duty cycle parameters ( $\tau_i$ ) are set to 0.5 so that the motors spend the same proportion of time in their two limit angles. 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.

**Random variation of controller’s parameters** Each parameter of the controller 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.

**Main Behavioral descriptor (duty factor)** The default 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

Leg number		1	2	3	4	5	6
First joint	$\alpha_{i_1}$	1.00	1.00	1.00	1.00	1.00	1.00
	$\phi_{i_1}$	0.00	0.00	0.50	0.50	0.00	0.00
	$\tau_{i_1}$	0.5	0.5	0.5	0.5	0.5	0.5
Two last joints	$\alpha_{i_2}$	0.25	0.25	0.25	0.25	0.25	0.25
	$\phi_{i_2}$	0.75	0.25	0.25	0.75	0.75	0.25
	$\tau_{i_2}$	0.5	0.5	0.5	0.5	0.5	0.5

Extended Data Table 1 | Parameters of the reference controller.

is in contact with the ground (1: contact, 0: no contact). The result is 6 Boolean time series ( $C_i$  for the  $i^{\text{th}}$  leg). The behavioral descriptor is then computed with the average of each time series:

$$\mathbf{x} = \begin{bmatrix} \frac{\sum_t C_1(t)}{\text{numTimesteps}(C_1)} \\ \vdots \\ \frac{\sum_t C_6(t)}{\text{numTimesteps}(C_6)} \end{bmatrix} \quad (6)$$

During the generation of the behavior-performance map, the behaviors are stored in the maps’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.

**Alternative Behavioral descriptor (orientation)** The alternative behavioral descriptor tested on the physical robot (we investigated many other descriptors in simulation: Supplementary Experiment S5) characterizes changes in the angular position of the robot during walking, measured as the proportion of 15ms intervals that each of the pitch, roll and yaw angles of the robot frame are positive (three dimensions) and negative (three additional dimensions):

$$\mathbf{x} = \begin{bmatrix} \frac{1}{K} \sum_k U(\Theta^T(k) - 0.005\pi) \\ \frac{1}{K} \sum_k U(-\Theta^T(k) - 0.005\pi) \\ \frac{1}{K} \sum_k U(\Psi^T(k) - 0.005\pi) \\ \frac{1}{K} \sum_k U(-\Psi^T(k) - 0.005\pi) \\ \frac{1}{K} \sum_k U(\Phi^T(k) - 0.005\pi) \\ \frac{1}{K} \sum_k U(-\Phi^T(k) - 0.005\pi) \end{bmatrix} \quad (7)$$

where  $\Theta^T(k)$ ,  $\Psi^T(k)$  and  $\Phi^T(k)$  denote the pitch, roll and yaw angles, respectively, of the robot torso (hence  $T$ ) at the end of interval  $k$ , and  $K$  denotes the number of 15ms intervals during the 5 seconds of simulated movement (here,  $K = 5s/0.015s \approx 334$ ). The unit step function  $U(\cdot)$  returns 1 if its argument exceeds 0, and returns 0 otherwise. To discount for insignificant motion around 0 rad, orientation angles are only defined as positive if they exceed 0.5% of  $\pi$  rad. Similarly, orientation angles are only defined as negative if they are less than  $-0.5\%$  of  $\pi$  rad.

**Performance function** In these experiments, the “mission” of the robot is to go forward as fast as possible. The performance of a controller, which is a set of parameters (section 1.4: Parametrized controller), is defined as how far the robot moves in a pre-specified direction in 5 seconds.

During the behavior-performance map creation step, the performance is obtained thanks to the simulation of the robot. All odometry results reported on the physical robot, during the adaptation step, are measured with the embedded simultaneous location and mapping (SLAM) algorithm<sup>46</sup>. 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 (Extended Data Fig. 9d).

Some damage to the robot may make it flip over. In such cases, the visual 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 on Supplementary Fig. 9d). In these cases the adaptation step algorithm will assume that the behavior is low-performing and will select another working behavior. Thus, the overall algorithm is not substantially impacted by such infrequent under-measurements of performance.

**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 map 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 map 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

$$\max(\mathbf{P}_{1:t}) \geq \alpha \max_{\mathbf{x} \in \mathcal{P}}(\mu_t(\mathbf{x})), \quad \text{with } \alpha = 0.9 \quad (8)$$

where  $\mathbf{x}$  is a location in the discrete behavioral space (i.e. a type of behavior) and  $\mu_t$  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 map, the algorithm terminates. At that point, the highest-performing solution found so far will be the compensatory behavior that the algorithm selects. An alternative way the algorithm can halt is if 20 tests on the physical robot occur without triggering the stopping criterion described in equation 8: 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. This user-defined stopping criterion is not strictly necessary, as the algorithm is guaranteed to stop in the worst case after every behavior in the map is tested, but it allows a practical limit on the number of trials performed on the physical robot.

**Initiating the Adaptation Step** The adaptation step is triggered when the performance drops by a certain amount. The simplest way to choose that threshold is to let the user specify it. Automating the selection of this value, and the impact of triggering the algorithm prematurely, is an interesting question for future research in this area.

#### Main parameters of MAP-Elites

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

- iterations: 40 million

### Main parameters of M-BOA

- $\sigma_{noise}^2$ : 0.001
- $\alpha$ : 0.9
- $\rho$ : 0.4
- $\kappa$ : 0.05

## 1.5 Robotic Arm Experiment

**Physical robot** The physical robot is a planar robotic arm with 8 degrees of freedom (Extended Data Fig. 7a) and a 1-degree-of-freedom gripper. The robot has to release a ball into a bin (a variant of the classic “pick and place” task in industrial robotics). To assess the position of the gripper, a red cap, placed on top of the gripper, is tracked with a video camera. The visual tracking is achieved with the “cmvision” ROS package, which tracks colored blobs (<http://wiki.ros.org/cmvision>). The eight joints of the robot are actuated by position-controlled servos manufactured by Dynamixel. To maximize the reliability of the arm, the type of servo is not the same for all the joints: heavy-duty servos are used near the base of the robot and lighter ones are used for the end of the arm. The first joint, fixed to the base, is moved by two MX-28 servos mounted in parallel. The second joint is moved by an MX-64 servo. The 3 subsequent servos are single MX-28s, and the 3 remaining servos are AX-18s. All the robot’s joints are limited to a motion range of  $\pm\pi/2$ .

**Simulator** The generation of the behavior-performance map is made with a simulated robot in the same way as for the hexapod experiment. For consistency with the simulated hexapod experiments, we used the dynamic (as opposed to kinematic) version of the simulator, based on the ODE library. Any joint configuration that resulted in the arm colliding with itself was not added to the map.

**Parametrized controller** The controller defines the target position for each joint. The controller is thus parametrized by eight continuous values from 0 to 1 describing the angle of each joint, which is mapped to the total motion range of each joint of  $\pm\pi/2$ . The 8 joints are activated simultaneously and are driven to their target position by internal PID controllers.

We chose this simple control strategy to make the experiments easy to reproduce and highlight the contribution of Intelligent Trial & Error for damage recovery. More advanced control strategies, for instance visual servoing<sup>7</sup>, would be more realistic in an industrial environment, but they would have made it hard to analyze the experimental results because both Intelligent Trial & Error and the controller would compensate for damage at the same time.

**Randomly varying the controller’s parameters** Each parameter of the controller (section “Parametrized controller”) has a 12.5% chance of being changed to any value from 0 to 1, with the new value chosen from a polynomial distribution as described on p. 124 of (Deb, 2000), with  $\eta_m = 10.0$ .

**Behavioral descriptor** Because the most important aspect of the robot’s behavior in this task is the final position of the gripper, we use it as the behavioral descriptor:

$$\text{behavioral\_descriptor}(\text{simu}(\mathbf{c})) = \begin{bmatrix} x_g \\ y_g \end{bmatrix} \quad (9)$$

where  $(x_g, y_g)$  denotes the position of the gripper once all the joint have reached their target position.

The size of the working area of the robot is a rectangle measuring  $1.4m \times 0.7m$ . For the behavior-performance map, this rectangle is discretized into a grid composed of 20000 square cells ( $200 \times 100$ ). The robot is 62cm long.

**Performance function** Contrary to the hexapod experiment, for the robotic arm experiment the performance function for the behavior-map creation step and for the adaptation step are different. We did so to demonstrate that the two can be different, and to create a behavior-performance map that would work with arbitrary locations of the target bin.

For the *behavior-performance map generation step* (accomplished via the MAP-Elites algorithm), the performance function captures the idea that all joints should contribute equally to the movement. Specifically, high-performance is defined as minimizing the variance of the joint angles, that is:

$$\text{performance}(\text{simu}(\mathbf{c})) = -\frac{1}{8} \sum_{i=0}^{i=7} (p_i - m)^2 \quad (10)$$

where  $p_i$  is the angular position of joint  $i$  (in radians) and  $m = \frac{1}{8} \sum_{i=0}^{i=7} p_i$  is the mean of the joint angles. This performance function does not depend on the target. The map is therefore generic: it contains a high-performing controller for each point of the robot’s working space.

For the *adaptation step* (accomplished via the M-BOA algorithm), the behavior-performance map, which is generic to many tasks, is used for a particular task. To do so, the adaptation step has a different performance measure than the step that creates the behavior-performance map. For this problem, the predicted performance measure is the Euclidean distance to the target (closer is better). Specifically, for each behavior descriptor  $\mathbf{x}$  in the map, performance is

$$\mathcal{P}(\mathbf{x}) = -\|\mathbf{x} - \mathbf{b}\| \quad (11)$$

where  $\mathbf{b}$  is the  $(x, y)$  position of the target bin. Note that the variance of the joint angles, which is used to create the behavior-performance map, is ignored during the adaptation step.

The performance of a controller on the physical robot is minimizing the Euclidean distance between the gripper (as measured with the external camera) and the target bin:

$$\text{performance}(\text{physical\_robot}(\mathcal{C}(\chi))) = -\|\mathbf{x}_g - \mathbf{b}\| \quad (12)$$

where  $\mathbf{x}_g$  is the position of the physical gripper after all joints have reached their final position,  $\mathbf{b}$  is the position of the bin, and  $\mathcal{C}(\chi)$  is the controller being evaluated ( $\chi$  is the position in simulation that controller reached).

If the gripper reaches a position outside of the working area, then the camera cannot see the marker. In these rare cases, we set the performance of the corresponding controller to a low value ( $-1m$ ).

For the control experiments with traditional Bayesian optimization on the physical robot (see Supplementary Experiment S1), self-collisions are frequent during adaptation, especially given that we initialize the process with purely random controllers (i.e. random joint angles). While a single self-collision is unlikely to break the robot, hundreds of them can wear out the gearboxes because each servo continues to apply a force for a period of time until it determines that it cannot move. To minimize costs, and because we ran 210 independent runs of the algorithm (14 scenarios  $\times$  15 replicates), we first tested each behavior in simulation (taking the damage into account) to check that there were no self-collisions. If we detected a self-collision, the performance for that behavior was set to a low value ( $-1m$ ).

Auto-collisions are much less likely with Intelligent Trial & Error because the behavior-performance map contains only controllers that do not self-collide on the undamaged, simulated

robot. As a consequence, in the Intelligent Trial & Error experiments we did not simulate controllers before testing them on the physical robot.

**Stopping criterion** Because the robot’s task is to release a ball into a bin, the adaptation step can be stopped when the gripper is above the bin. The bin is circular with a diameter of 10 cm, so we stopped the adaptation step when the red cap is within 5 cm of the center of the bin.

**Main MAP-Elites parameters for the robotic arm experiment:**

- parameters in controller: 8
- controller parameter values: 0 to 1 (continuous)
- dimensions in the behavioral space: 2
- simulated evaluations to create the behavior-performance map: 20 million

**Main M-BOA parameters for the robotic arm experiment:**

- $\sigma_{noise}^2$ : 0.03
- $\rho$ : 0.1
- $\kappa$ : 0.3

## 1.6 Selection of parameters

All of the data reported in this section comes from experiments with the simulated hexapod robot, unless otherwise stated.

**Selecting the  $\rho$  value** For  $\rho$  between 0.1 and 0.8, we counted the number of behaviors from the map that would be influenced by a single test on the real hexapod robot (we considered a behavior to be influenced when its predicted performance was affected by more than 25% of the magnitude of the update for the tested behavior): with  $\rho = 0.2$ , the update process does not affect any neighbor in the map, with  $\rho = 0.4$ , it affects 10% of the behaviors, and with  $\rho = 0.8$ , it affects 80% of them. Additional values are shown in Extended Data Fig. 9c.

The previous paragraph describes tests we conducted to determine the number of behaviors in the map affected by different  $\rho$  values, but those experiments do not tell us how different  $\rho$  values affect the performance of the algorithm overall. To assess that, 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 independently generated replicates of the default 6-dimensional behavior-performance map. The algorithm stopped if 20 adaptation iterations passed without success according to the stopping criteria described in the main text and section 1.4: Stopping criterion. The results reveal that median performance decreases only modestly, but significantly, when the value of  $\rho$  increases: changing  $\rho$  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, Extended Data Fig. 9b). 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 (Extended Data Fig. 9b, dotted red line) is higher than the minimum performance for more extreme values of  $\rho$ .

A larger effect of changing  $\rho$  is the amount of time required to find a compensatory behavior, which decreases when the value of  $\rho$  increases (Extended Data Fig. 9b). With a  $\rho$  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  $\rho$  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 hexapod experiments because it represents a good trade-off between a high minimum performance and a low number of physical tests on the robot. The value of  $\rho$  for the robotic arm experiment has been chosen with the same method.

**Selection of the  $\kappa$  value** For the hexapod robot experiments, 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 behavior-performance map creation step: the map 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.

For the robotic arm experiments, we chose  $\kappa = 0.3$ , which emphasizes exploration more, because it experimentally leads to better results.

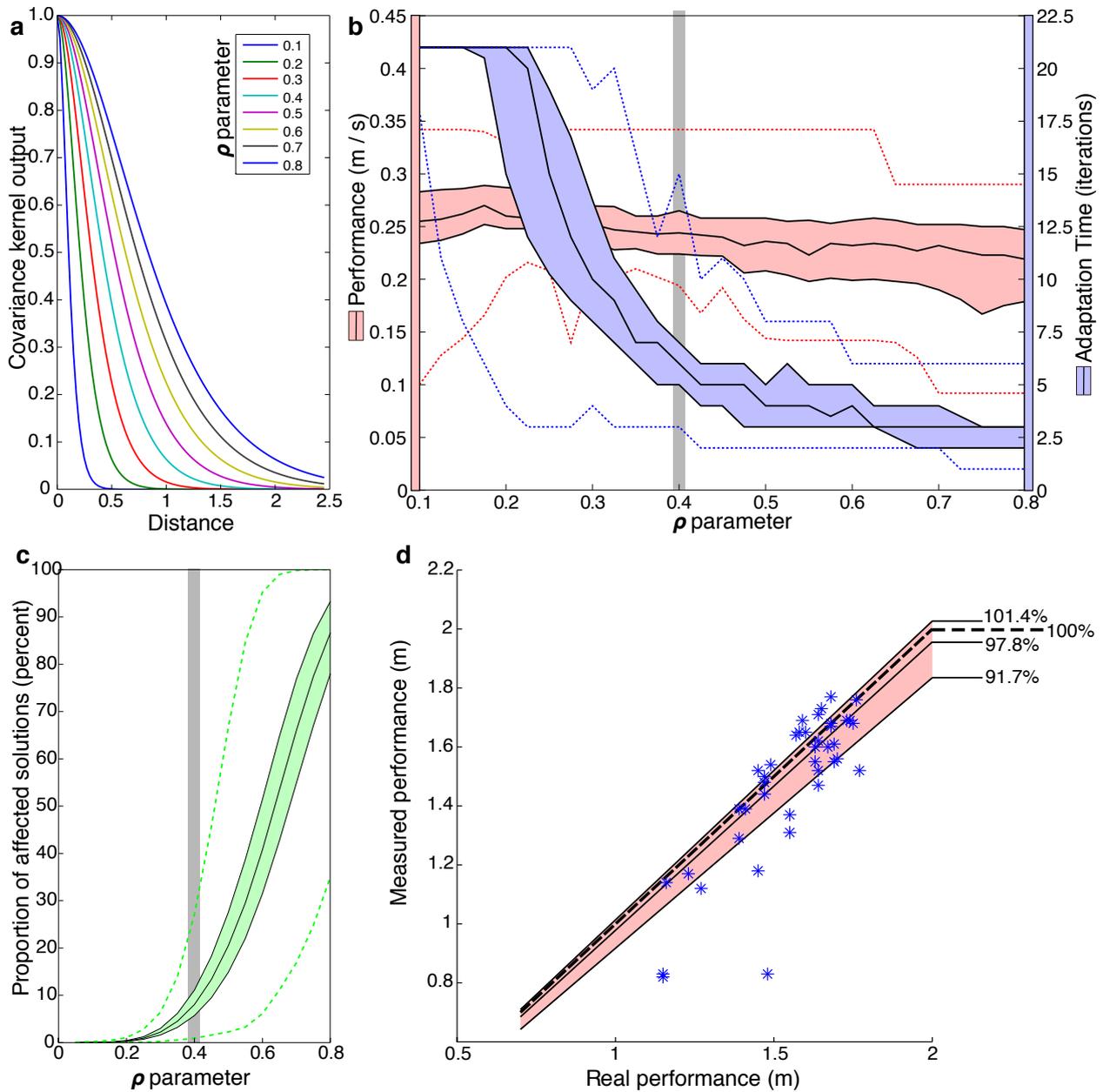
## 1.7 Running time

**Computing hardware** All computation (on the physical robots 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 behavior-performance map creation step. Creating one map for the hexapod experiment took 2 weeks, taking advantage of the fact that map creation can easily be parallelized across multiple cores. Map 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 map.

For the hexapod robot experiment, the most expensive part of adaptation is the Simultaneous Localization And Mapping (SLAM) algorithm<sup>46,59,60</sup>, which measures the distance traveled on the physical hexapod 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 adaptation 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.

**Measuring how long adaptation takes (hexapod robot)** 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 hexapod robot takes about 8 seconds (median 8.03 seconds, 5<sup>th</sup> and 95<sup>th</sup> 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 because the size of the Kernel Matrix (K matrix, see Methods and Extended Data Fig. 8), 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<sup>th</sup> selection takes 0.31 [0.17; 0.34] seconds.



**Extended Data Figure 9 | The effect of changing the algorithm's parameters.** (a) The shape of the Matérn kernel function for different values of the  $\rho$  parameter. (b) Performance and required adaptation time obtained for different values of  $\rho$ . For each  $\rho$  value, the M-BOA algorithm was executed in simulation with 8 independently generated behavior-performance maps and for 6 different damage conditions (each case where one leg is missing). (c) The number of controllers in the map affected by a new observation according to different values of the  $\rho$  parameter. (d) The precision of the odometry value. The distances traveled by the physical robot, as measured manually ("real performance") is compared to the measurements automatically provided by the simultaneous location and mapping (SLAM) algorithm ("measured performance"). The dashed black line indicates the hypothetical case where SLAM measurements are error-free and thus are the same as manual measurements. In (b), (c) and (d), the middle, black lines represent medians and the borders of the shaded areas show the 25<sup>th</sup> and 75<sup>th</sup> percentiles. The dotted lines are the minimum and maximum values. The gray bars show the  $\rho$  value chosen for the hexapod experiments in the main text.

## 2 Supplementary Experiments S1

### Additional conditions for the robotic arm

**Methods** We investigated 11 damage conditions on the physical robot in addition to the 3 described in the main text (Fig. 3). We used the same setup as described in the main text (see main text and section 1.5). Extended Data Fig. 7 shows the 14 scenarios.

For each of the 14 damage scenarios, we replicated experiments on the physical robot with 15 independently generated behavior-performance maps (210 runs in total). We also replicated control experiments, which consist of traditional Bayesian optimization directly in the original parameter space (i.e. without behavior-performance maps), 15 times for each of the 14 damage conditions (210 runs in total). For both the experimental and control treatments, each experiment involved 30 evaluations on the physical robot (31 if the first trial is counted). In many cases, not all 30 evaluations were required to reach the target, so we report only the number of trials required to reach that goal.

**Results** After running the MAP-Elites algorithm for 20 million evaluations, each of the 15 generated maps contain more than 11,000 behaviors (11,209 [1,1206; 1,1217] behaviors, Extended Data Fig. 7c).

In all the generated maps, the regions of different performance values for behaviors are arranged in concentric shapes resembling cardioids (inverted, heart-shaped curves) that cover the places the robot can reach (Extended Data Fig. 7c). The black line drawn over the shown map corresponds to all the positions of the end-effector for which all the degrees of freedom are set to the same angle (from  $-\pi/4$  to  $+\pi/4$ ), that is, for the theoretically highest achievable performance (i.e. the lowest possible variance in servo angles). The performance of the behaviors tends to decrease the further they are from this optimal line.

The adaptation results (Extended Data Fig. 7e) show that the Intelligent trial and error algorithm manages to reach the goal of being less than 5 cm from the center of the bin for all the runs in all the tested scenarios save two (scenarios 11 & 12). For these two scenarios, the algorithm still reaches the target 60% and 80% of the time, respectively. For all the damage conditions, the Intelligent Trial and Error algorithm reaches the target significantly more often than the Bayesian optimization algorithm ( $p < 10^{-24}$ ). Specifically, the median number of iterations to reach the target (Extended Data Fig. 7f) is below 11 iterations (27.5 seconds) for all scenarios except 11 and 12, for which 31 and 20 iterations are required, respectively. When the robot is not able to reach the target, the recorded number of iterations is set to 31, which explains why the median number of iterations for the Bayesian optimization algorithm is equal to 31 for most damage conditions. For all the damage conditions except one (scenario 11), the Intelligent Trial and Error algorithm used fewer trials to reach the target than the traditional Bayesian optimization algorithm.

If the robot is allowed to continue its experiment after reaching the 5 cm radius tolerance, for a total of 31 iterations (Extended Data Fig. 7g), it reaches an accuracy around 1 cm for all the damage conditions except the two difficult ones (scenarios 11 and 12). This level of accuracy is never achieved with the classic Bayesian optimization algorithm, whose lowest median accuracy is 2.6cm.

Scenarios 11 and 12 appear to challenge the Intelligent Trial and Error algorithm. While in both cases the success rate is improved, though not substantially, in case 11 the median accuracy is actually lower. These results stem from the fact that the difference between the successful pre-damage and post-damage behaviors is so large that the post-damage solutions for both scenarios lie outside of the map. This illustrates a limit of the proposed approach: if the map does not contain a behavior able to cope

with the damage, the robot will not be able to adapt. This limit mainly comes from the behavioral descriptor choice: we chose it because of its simplicity, but it does not capture all of the important dimensions of variation of the robot. More sophisticated descriptors are likely to allow the algorithm to cope with such situations. On the other hand, this experiment shows that with a very simple behavioral descriptor, using only the final position of the end-effector, our approach is able to deal with a large variety of different target positions and is significantly faster than the traditional Bayesian optimization approach (Extended Data Fig. 7d, maximum p-value over each time step  $< 10^{-16}$ ), which is the current state of the art technique for direct policy search in robotics<sup>32,61,42,17</sup>.

## 3 Supplementary Experiments S2

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

**Methods** The Intelligent Trial and Error Algorithm relies on three main concepts: (1) the creation of a behavior-performance map in simulation via the MAP-Elites algorithm, (2) searching this map 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 behavior-performance map 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: in each of them, we deactivated one of these three subcomponents or replaced it with an alternative algorithm from the literature. We then tested these variants on the hexapod robot. The variants are as follows:

- Variant 1 (MAP-Elites in 6 dimensions + random search): evaluates the benefit of searching the map via Bayesian optimization by searching that map with random search instead. Each iteration, a behavior is randomly selected from the map 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 behavior-performance map. In this variant, the Gaussian process is initialized with a constant mean (the average performance of the map: 0.24 m/s) at each location in the behavior space and a constant variance (the average variance of the map's performance:  $0.005 m^2/s^2$ ). As is customary, the first few trials (here, 5) of the Bayesian optimization process are selected randomly instead of letting the algorithm choose those points, which is known to improve performance.<sup>42</sup>
- Variant 3 (MAP-Elites in 6 dimensions + policy gradient): evaluates the benefit of Bayesian optimization compared to a more classic, local search algorithm<sup>17,62</sup>; 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 map 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 map 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<sup>32,42</sup>. In this variant, the Gaussian process is initialized with a constant mean set to

zero and with a constant variance ( $0.002m^2/s^2$ ). As described above, the five first trials are selected from pure random search to prime the Bayesian optimization algorithm<sup>42</sup>.

- Variant 5 (Policy gradient in the original parameter space of 36 dimensions): a stochastic gradient descent in the original parameter space<sup>62</sup>. This approach is a classic reinforcement learning algorithm for locomotion<sup>17</sup> and it is a baseline in many papers<sup>32</sup>.

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 1.4: Simulator) to emulate 6 different possible damage conditions, each of which involved removing a different leg. For variants in which MAP-Elites creates a map (variants 1, 2 and 3), we used the same maps from the main experiments (the eight independently generated maps, 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 maps 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 17 trials, which was the maximum number of iterations used by the Intelligent Trial and Error Algorithm, and the second case is after 150 trials, which is approximately the number of trials used in previous work<sup>62,32,42</sup>.

**Results** After 17 trials on the robot, Intelligent Trial and Error significantly outperforms all the variants (Extended Data Fig. 2b,  $p < 10^{-67}$ , Intelligent Trial and Error performance: 0.26 [0.20; 0.33] 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 map performs the best (Variant 1: 0.21 [0.16; 0.27] m/s), followed by Bayesian optimization in the map (Variant 2: 0.20 [0.13; 0.25] m/s), and policy gradient in the map (Variant 3: 0.13 [0; 0.23] m/s). Variants that search directly in the parameter space did not find any working behavior (Variant 4, Bayesian optimization: 0.04m/s, [0.01; 0.09]; 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 map only contains high-performing solutions, which are the result of the intense search of the MAP-Elites algorithm (40 million evaluations in simulation). The map 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. 17 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<sup>62</sup>, the policy gradient algorithm performs 15 trials at each iteration. Consequently, when only 17 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 (Extended Data Fig. 2b).

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 map. As mentioned previously, no working controller was found in the two variants directly searching in this high-dimensional space.

Overall, the analysis after 17 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 map of high-performing behaviors in that space:  $p < 5 \times 10^{-50}$  when comparing variants searching in the behavior-performance map 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 behavior-performance map creation step (with initialization: 0.26 [0.20; 0.33] m/s, without initialization: 0.20 [0.13; 0.25] m/s,  $p = 10^{-96}$ ).

To check whether these variants might perform better if allowed the number of evaluations typically given to previous state-of-the-art algorithms<sup>62,32,42</sup>, we continued the experiments until 150 trials on the robot were conducted (Extended Data Fig. 2c). Although the results for all the variants improved, Intelligent Trial and Error still outperforms all them ( $p < 10^{-94}$ ; Intelligent Trial and Error: 0.31 [0.26; 0.37] m/s, random search: 0.26 [0.22; 0.30] m/s, Bayesian optimization: 0.25 [0.18; 0.31] m/s, policy search: 0.23 [0.19; 0.29] m/s). These results are consistent with the previously published results<sup>62,32,42,17</sup>, 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 evaluations, Bayesian optimization and policy gradient both perform much worse (Bayesian optimization: 0.08 [0.05; 0.12]; policy gradient: 0.06 [0.01; 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 for both damage recovery<sup>63,12,64,65,66</sup> and gait learning<sup>67,62,68,48,69,32,61,42,17,49</sup>, and can therefore be considered the state of the art.

## 4 Supplementary Experiments S3

### Robustness to environmental changes

**Methods** The map 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 map 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 (Extended Data Fig. 3a). We launched 10 replicates for each of the eight maps and each one-degree increment between  $-20^\circ$  and  $+20^\circ$ , for a total of  $10 \times 8 \times 41 = 3280$  experiments. As in

Supplementary Experiments S2, 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.

**Results** The results show that, when the slope is negative (descending), the Intelligent Trial and Error approach finds fast gaits in fewer than 3 trials. For reference, a hand-designed, classic, tripod gait (section 1.4) falls on slopes below  $-15^\circ$  degrees. When the slope is positive (ascent), Intelligent Trial and Error finds slower behaviors, as is expected, but even above  $10^\circ$  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 with the same setup as Experiments S2 (6 damage conditions). We launched 10 replicates for each damage condition, for 8 independently generated behavior-performance maps, and each two-degree increment between  $-20^\circ$  and  $+20^\circ$  degrees. There are therefore 480 replicates for each two-degree increment between  $-20^\circ$  and  $+20^\circ$ , for a total of  $480 \times 21 = 10080$  experiments.

Intelligent Trial and Error is not critically affected by variations of slope between  $-10^\circ$  and  $+10^\circ$  (Extended Data Fig. 3b): 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 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  $-10^\circ$  and above  $10^\circ$ , 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.

## 5 Supplementary Experiments S4

### Comparison between MAP-Elites and Random Sampling

**Methods** The MAP-Elites algorithm is a stochastic search algorithm that attempts to fill a discretized map with the highest-

performing solution at each point in the map. As explained in the main text, each point in the map represents a different type of behavior, as defined by the behavioral dimension of the map. MAP-Elites generates new candidate points by randomly selecting a location in the map, changing the parameters of the controller that is stored there, and then saving that controller in the appropriate map location if it is better than the current occupant at that location. Intuitively, generating new candidate solutions from the best solutions found so far should be better than generating a multitude of controllers randomly and then keeping the best one found for each location in the map. In this section we report on experiments that confirm that intuition.

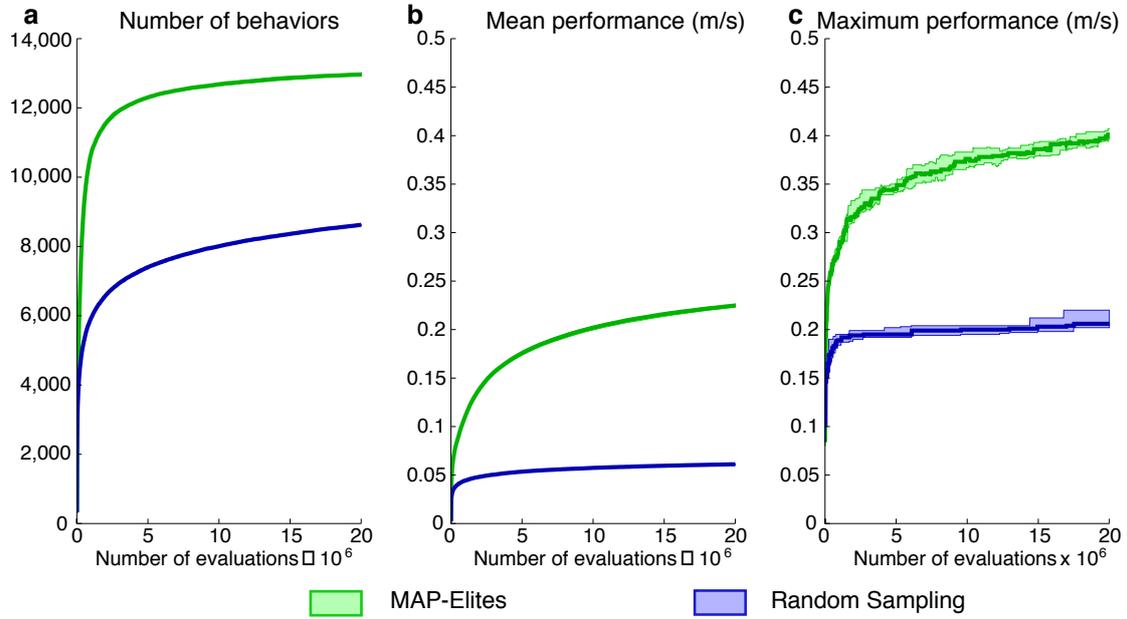
To understand the advantages of MAP-Elites over random sampling, we compared the two algorithms by generating data with the simulated hexapod. The experiments have the same virtual robot, environment, controller, performance function, and behavioral descriptors as in the main experiments (see Methods). We analyzed the number of cells for which a solution is found (an indication of the diversity of behavior types the algorithms generate), the average performance of behaviors in the map, and the maximum performance discovered.

We replicated each experiment 8 times, each of which included 20 million evaluations on the simulated robot.

**Results** The results show that the MAP-Elites algorithm outperforms random sampling on each of these measures (Extended Data Fig. 10). After 20 million evaluations, about 13000 cells (median: 12968, 5<sup>th</sup> & 95<sup>th</sup> percentiles: [12892; 13018]) are filled by MAP-Elites (about 83% percent of the map), whereas random sampling only filled approximately 8600 (8624 [8566; 8641]) cells (about 55% percent of the map) (Extended Data Fig. 10a). The difference between the two algorithms is large and appears early (Extended Data Fig. 10a); after only 1 million evaluations, MAP-Elites filled 10670 [10511; 10775] cells (68% of the map), whereas random sampling filled 5928 [5882; 5966] cells (38% of the map).

The solutions discovered by MAP-Elites are not only more numerous, but also outperform those found by random sampling (Extended Data Fig. 10b): with MAP-Elites, after 20 million evaluations the average performance of filled cells is 0.22 [0.22; 0.23] m/s, whereas it is 0.06 [0.06; 0.06] m/s with random sampling, which is similar to the performance obtained with the reference controller on a damaged robot (Fig. 3). These two results demonstrate that MAP-Elites is a much better algorithm than random sampling to find a map of the diverse, "elite" performers in a search space.

In addition, MAP-Elites is a better optimization algorithm, as measured by the performance of the best single solution produced. The performance of the best solution in the map after 20 million evaluations is 0.40 [0.39; 0.41] m/s with MAP-Elites, compared to 0.21 [0.20; 0.22] m/s with random sampling.



Extended Data Figure 10 | Comparing MAP-Elites and random sampling for generating behavior-performance maps. (a) The number of points in the map for which a behavior is discovered. (b) The mean performance of the behaviors in the map. (c) The maximum performance of the behaviors in the map. For all these figures, the middle lines represent medians over 8 independently generated maps and the shaded regions extend to the 25<sup>th</sup> and 75<sup>th</sup> percentiles, even for (a) and (b), where the variance of the distribution is so small that it is difficult to see. See Supplementary Experiment S4 for methods and analysis.

## 6 Supplementary Experiments S5

### Alternative behavioral descriptors

**Methods** To create a map with MAP-Elites, one has to define the dimensions of the behavioral space, i.e. the behavioral descriptors. The main experiments show that using a predefined behavioral descriptor (the proportion of time that each leg of a hexapod robot is in contact with the ground, i.e. the duty factor) creates a map that provides useful priors for damage recovery.

This section describes how we tested (in simulation) how performance is affected by alternative behavioral descriptors, including descriptors that have a different number of dimensions. We also evaluated how performance is affected if the behavioral descriptors are randomly selected from a large list of potential descriptors. This test simulates the algorithm’s performance if the behavioral descriptors are chosen without insight into the problem domain.

The behavioral descriptors we tested are as follows:

1. Duty factor (6-dimensional): This descriptor is the default one from the main experiment. It corresponds to the proportion of time each leg is in contact with the ground:

$$\mathbf{x} = \begin{bmatrix} \frac{\sum_t C_1(t)}{\text{numTimesteps}} \\ \vdots \\ \frac{\sum_t C_6(t)}{\text{numTimesteps}} \end{bmatrix} \quad (13)$$

where  $C_i(t)$  denotes the Boolean value of whether leg  $i$  is in contact with the ground at time  $t$  (1: contact, 0: no contact).

2. Orientation (6-dimensional): This behavioral descriptor characterizes changes in the angular position of the robot during walking, measured as the proportion of 15ms intervals that each of the pitch, roll and yaw angles of the robot frame are positive (three dimensions) and negative (three additional dimensions):

$$\mathbf{x} = \begin{bmatrix} \frac{1}{K} \sum_k U(\Theta^T(k) - 0.005\pi) \\ \frac{1}{K} \sum_k U(-\Theta^T(k) - 0.005\pi) \\ \frac{1}{K} \sum_k U(\Psi^T(k) - 0.005\pi) \\ \frac{1}{K} \sum_k U(-\Psi^T(k) - 0.005\pi) \\ \frac{1}{K} \sum_k U(\Phi^T(k) - 0.005\pi) \\ \frac{1}{K} \sum_k U(-\Phi^T(k) - 0.005\pi) \end{bmatrix} \quad (14)$$

where  $\Theta^T(k)$ ,  $\Psi^T(k)$  and  $\Phi^T(k)$  denote the pitch, roll and yaw angles, respectively, of the robot torso (hence  $T$ ) at the end of interval  $k$ , and  $K$  denotes the number of 15ms intervals during the 5 seconds of simulated movement (here,  $K = 5s/0.015s \approx 334$ ). The unit step function  $U(\cdot)$  returns 1 if its argument exceeds 0, and returns 0 otherwise. To discount for insignificant motion around 0 rad, orientation angles are only defined as positive if they exceed 0.5% of  $\pi$  rad. Similarly, orientation angles are only defined as negative if they are less than  $-0.5\%$  of  $\pi$  rad.

3. Displacement (6-dimensional): This behavioral descriptor characterizes changes in the position of the robot during walking. It is measured as the proportion of 15ms intervals that the robot is positively or negatively displaced along each of the  $x$ ,  $y$ , and  $z$  axes:

$$\mathbf{x} = \begin{bmatrix} \frac{1}{K} \sum_k U(\Delta x(k) - 0.001) \\ \frac{1}{K} \sum_k U(-\Delta x(k) - 0.001) \\ \frac{1}{K} \sum_k U(\Delta y(k) - 0.001) \\ \frac{1}{K} \sum_k U(-\Delta y(k) - 0.001) \\ \frac{1}{K} \sum_k U(\Delta z(k) - 0.001) \\ \frac{1}{K} \sum_k U(-\Delta z(k) - 0.001) \end{bmatrix} \quad (15)$$

where  $[\Delta x(k), \Delta y(k), \Delta z(k)]$  denote the linear displacement in meters of the robot during interval  $k$ , and  $K$  denotes the number of 15ms intervals during 5 seconds of simulated movement (here,  $K = 5s/0.015s \approx 334$ ). The unit step function  $U(\cdot)$  returns a value of 1 if its argument exceeds 0, and returns a value of 0 otherwise. To ignore insignificant motion, linear displacements are defined to be positive if they exceed 1mm, and are defined to be negative if they are less than  $-1\text{mm}$ .

4. Total energy expended per leg (6-dimensional): This behavioral descriptor captures the total amount of energy expended to move each leg during 5 seconds of movement:

$$\mathbf{x} = \begin{bmatrix} \frac{E_1}{M_E} \\ \vdots \\ \frac{E_6}{M_E} \end{bmatrix} \quad (16)$$

where  $E_i$  denotes the energy utilized by leg  $i$  of the robot during 5 seconds of simulated movement, measured in N.m.rad.  $M_E$  is the maximum amount of energy available according to the servo model of the simulator, which for 5 seconds is 100 N.m.rad.

5. Relative energy expended per leg (6-dimensional): This behavioral descriptor captures the amount of energy expended to move each leg relative to the energy expended by all the legs during 5 seconds of simulated movement:

$$\mathbf{x} = \begin{bmatrix} \frac{E_1}{\sum_{i=1..6} E_i} \\ \vdots \\ \frac{E_6}{\sum_{i=1..6} E_i} \end{bmatrix} \quad (17)$$

where  $E_i$  denotes the energy utilized by leg  $i$  of the robot during 5 seconds of simulated movement, measured in N.m.rad.

6. Deviation (3-dimensional): This descriptor captures the range of deviation of the center of the robot frame versus the expected location of the robot if it traveled in a straight line at a constant speed.

$$\mathbf{x} = \begin{bmatrix} \frac{0.95 \left( \max_t(x(t)) - \min_t(x(t)) \right)}{0.2} \\ \frac{0.95 \left( \max_t(y(t) - \frac{y_{\text{final}}}{5} \times t) - \min_t(y(t) - \frac{y_{\text{final}}}{5} \times t) \right)}{0.2} \\ \frac{0.95 \left( \max_t(z(t)) - \min_t(z(t)) \right)}{0.2} \end{bmatrix} \quad (18)$$

where  $[x(t), y(t), z(t)]$  denote the position of robot's center at time  $t$ , and  $[x_{\text{final}}, y_{\text{final}}, z_{\text{final}}]$  denote its final position after 5 seconds.

The robot's task is to move along the  $y$ -axis. Its starting position is  $(0,0,0)$ . The deviation along the  $x$  and  $z$  axes is computed as the maximum difference in the robot's position in those dimensions at any point during 5 seconds. For the  $y$  axis,  $\frac{y_{\text{final}}}{5}$  corresponds to the average speed of the robot (the distance covered divided by total time), therefore  $\frac{y_{\text{final}}}{5} \times t$  is the expected position at timestep  $t$  if the robot was moving at constant speed. The deviation from the  $y$  axis is computed with respect to this "theoretical" position.

To obtain values in the range  $[0,1]$ , the final behavioral descriptors are multiplied by 0.95 and then divided by 20 cm (these values were determined empirically).

7. Total ground reaction force per leg (6-dimensional): This behavioral descriptor corresponds to the amount of force each leg applies to the ground, measured as a fraction the total possible amount of force that a leg could apply to the ground. Specifically, the measurement is

$$\mathbf{x} = \begin{bmatrix} \frac{F_1}{M_F} \\ \vdots \\ \frac{F_6}{M_F} \end{bmatrix} \quad (19)$$

where  $F_i$  denotes the ground reaction force (GRF) each leg  $i$  generates, averaged over 5 seconds of simulated movement.  $M_F$  is the maximum such force that each leg can apply, which is 10N.

8. Relative ground reaction force per leg (6-dimensional): This behavioral descriptor corresponds to the amount of force each leg applies to the ground, relative to that of all the legs:

$$\mathbf{x} = \begin{bmatrix} \frac{F_1}{\sum_{i=1..6} F_i} \\ \vdots \\ \frac{F_6}{\sum_{i=1..6} F_i} \end{bmatrix} \quad (20)$$

where  $F_i$  denotes the ground reaction force (GRF) each leg  $i$  generates, averaged over 5 seconds of simulated movement.

9. Lower-leg pitch angle (6-dimensional): This descriptor captures the pitch angle for the lower-leg with respect to the ground (in a global coordinate frame), averaged over 5 seconds:

$$\mathbf{x} = \begin{bmatrix} \frac{\sum_t \Theta_1^L(t)}{\pi \times N_1} \\ \vdots \\ \frac{\sum_t \Theta_6^L(t)}{\pi \times N_6} \end{bmatrix} \quad (21)$$

where  $\Theta_i^L(t)$  is the pitch angle of lower-leg  $i$  (hence the  $L$  in  $\Theta_i^L$ ) when it is in contact with the ground at time  $t$ , and  $N_i$  is the number of time-steps for which lower-leg  $i$  touches the ground. The foot pitch angles are in range  $[0, \pi]$  (as the leg can not penetrate the ground) and normalized to  $[0, 1]$ .

10. Lower-leg roll angle (6-dimensional): This descriptor captures the roll angle for the lower-leg with respect to the ground (in a global coordinate frame), averaged over 5 seconds:

$$\mathbf{x} = \begin{bmatrix} \frac{\sum_t \Psi_1^L(t)}{\pi \times N_1} \\ \vdots \\ \frac{\sum_t \Psi_6^L(t)}{\pi \times N_6} \end{bmatrix} \quad (22)$$

where  $\Psi_i^L(t)$  is the roll angle of lower-leg  $i$  (hence  $L$  in  $\Psi_i^L$ ) when it is in contact with the ground at time  $t$ , and  $N_i$  is the number of time-steps for which lower-leg  $i$  touches the ground. The foot roll angles are in range  $[0, \pi]$  (as the leg can not penetrate the ground) and normalized to  $[0, 1]$ .

11. Lower-leg yaw angle (6-dimensional): This descriptor captures the yaw angle for the lower-leg with respect to the ground (in a global coordinate frame), averaged over 5 seconds:

$$\mathbf{x} = \begin{bmatrix} \frac{\sum_t \Phi_1^L(t) + \pi}{2\pi \times N_1} \\ \vdots \\ \frac{\sum_t \Phi_6^L(t) + \pi}{2\pi \times N_6} \end{bmatrix} \quad (23)$$

where  $\Phi_i^L(t)$  is the yaw angle of lower-leg  $i$  (hence  $L$  in  $\Phi_i^L$ ) when it is in contact with the ground at time  $t$ , and  $N_i$  is the number of time-steps for which lower-leg  $i$  touches the ground. The foot yaw angles are in range  $[-\pi, \pi]$  and are normalized to  $[0, 1]$ .

12. Random (6-dimensional): The random behavioral descriptor differs from the other intentionally chosen descriptors in that it does not consist of one type of knowledge, but is instead randomly selected as a subset of variables from the previously described 11 behavioral descriptors. This descriptor is intended to simulate a situation in which one has little expectation for which behavioral descriptor will perform well, so one quickly picks a few different descriptor dimensions without consideration or experimentation. Instead of generating one such list in this fashion, we randomly sample from a large set to find the average performance of this approach over many different possible choices. For the random descriptor, each of the 6-dimensions is selected at random (without replacement) from the  $1 \times 3 + 10 \times 6 = 63$  available behavior descriptor dimensions described in the previous 11 descriptors (1 of the above descriptors is three-dimensional and the other 10 are six-dimensional):

$$\mathbf{x} = \begin{bmatrix} R_1 \\ \vdots \\ R_6 \end{bmatrix} \quad (24)$$

where  $R_i$  denotes the  $i^{\text{th}}$  dimension of the descriptor, randomly selected uniformly and without replacement from the 63 available dimensions in behavior descriptors 1-11.

It was necessary to compare these behavioral descriptors 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 1.4) to emulate 6 different possible damage conditions, each of which involved removing a different leg. The MAP-Elites algorithm, run for 3 million iterations, was used to create the behavior-performance maps for each of the behavioral descriptors (using a simulation of the undamaged robot). During the generation of the behavior-performance maps, the behaviors were stored in the map’s cells by discretizing each dimension of the behavioral descriptor space with these five values:  $\{0, 0.25, 0.5, 0.75, 1\}$  for the 6-dimensional behavioral descriptors, and with twenty equidistant values between  $[0, 1]$  for the 3-dimensional behavioral descriptor. During the adaptation phase, the behaviors were used with their actual values and thus not discretized.

We independently generated eight maps for each of the 11 intentionally chosen behavioral descriptors. Twenty independently generated maps were generated for the random behavioral descriptor. We launched ten replicates of each descriptor for each of the maps (eight for intentionally chosen behavioral descriptors and twenty for random behavioral descriptor) and each of the six damage conditions. There are therefore  $10 \times 8 \times 6 = 480$  replicates for each of the intentionally chosen descriptors, and  $10 \times 20 \times 6 = 1200$  replicates for the random descriptor. 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 behavioral descriptor after two different numbers of trials: the first case is after 17 trials, and the second case is after 150 trials.

**Results** The following results include 17 trials on the simulated robot, which was the maximum number of trials required for Intelligent Trial and Error to find a compensatory gait in the Supplementary Experiment S2. The post-adaptation performance

achieved with our alternative, intentionally chosen behavioral descriptors (numbers 2-11) was similar to the original duty factor behavioral descriptor (number 1) (Extended Data Fig. 4a). All 11 alternative, intentionally chosen descriptors (numbers 2-11) led to a median performance within 17% of the duty factor descriptor (performance: 0.241 [0.19; 0.29] m/s). The difference in performance was effectively nonexistent with the deviation descriptor (0.241 [0.14; 0.31] m/s), the total GRF descriptor (0.237 [0.15; 0.30] m/s), and the lower-leg roll angle descriptor (0.235 [0.14; 0.31] m/s). The lowest performance was discovered with the relative GRF descriptor (16.7% lower than the duty factor descriptor, 0.204 [0.08; 0.31] m/s). In terms of statistical significance, the performance achieved with the duty factor descriptor was no different from the deviation ( $p = 0.53$ ) and total GRF ( $p = 0.29$ ) descriptors. With all the remaining descriptors, the difference in performance was statistically significant ( $p < 10^{-3}$ ), but it did not exceed 0.04m/s. Additionally, the compensatory behaviors discovered with all our 11 alternative, intentionally chosen descriptors were always faster than the reference gait for all damage conditions.

To check whether our alternative, intentionally chosen behavioral descriptors lead to better performance if allowed a higher number of evaluations, we extended the experiments to 150 trials on the robot (Extended Data Fig. 4b). After 150 trials, the difference in performance between the duty factor behavioral descriptor (0.277 [0.24; 0.34] m/s) and our alternative behavioral descriptors was further reduced. For all but three alternative, intentionally chosen descriptors (displacement, total GRF and lower-leg yaw angle), the median performance was within 4% of the duty factor descriptor. The difference in performance was at  $\pm 3.6\%$  with the orientation (0.274 [0.22; 0.32] m/s), total energy (0.274 [0.19; 0.33] m/s), relative energy (0.273 [0.20; 0.32] m/s), deviation (0.287 [0.21; 0.34] m/s), relative GRF (0.266 [0.15; 0.35] m/s), lower-leg pitch angle (0.271 [0.21; 0.34] m/s) and lower-leg roll angle (0.268 [0.17; 0.34] m/s) descriptors. In the three remaining behavioral descriptors, displacement, total GRF, and lower-leg yaw angle, the performance was 0.264 [0.18; 0.32] m/s, 0.299 [0.25; 0.35] m/s and 0.255 [0.18; 0.32] m/s, respectively (difference at  $\pm 7.8\%$  of duty factor descriptor in all three cases). In terms of statistical significance, the performance achieved with the duty factor descriptor was barely statistically significantly different from the deviation descriptor ( $p = 0.041$ ). In all the remaining descriptors, the performance difference was statistically significant ( $p < 10^{-2}$ ), but no larger than 0.02m/s.

Our random behavioral descriptor also performed similarly to the duty factor descriptor. After 17 trials, the performance of M-BOA with the maps generated by the random descriptor was 0.232 [0.14; 0.30] m/s (4.2% lower than the duty factor descriptor performance). While the difference is statistically significant ( $p < 10^{-3}$ ), the difference in performance itself was negligible at 0.01m/s. This difference in performance was further reduced to 3.6% after 150 trials (random descriptor performance: 0.274 [0.21; 0.34] m/s, duty factor description performance: 0.277 [0.24; 0.34] m/s,  $p = 0.002$ ). Moreover, as with the intentionally chosen behavioral descriptors, the compensatory behavior discovered with the random descriptor was also faster than the reference gait.

These experiments show that the selection of the behavioral dimensions is not critical to get good results. Indeed, all tested behavioral descriptors, even those randomly generated, performed well (median  $> 0.20$  m/s after 17 trials). On the other hand, if the robot’s designers have some prior knowledge about which dimensions of variation are likely to reveal different types of behaviors, the algorithm can benefit from this knowledge to further improve results (as with the duty factor descriptor).

## 7 Caption for Supplementary Videos

### Video S1

This video can be viewed at: <https://youtu.be/T-c17RKh3uE>

#### Damage Recovery in Robots via Intelligent Trial and Error.

The video shows the Intelligent Trial and Error Algorithm in action with the two robots introduced in this paper: the hexapod robot and the 8 degrees of freedom robotic arm (Fig. 3). The video shows several examples of the different types of behaviors that are produced during the behavior-performance map creation step, from classic hexapod gaits to more unexpected forms of locomotion. Then, it shows how the hexapod robot uses that behavior-performance map to deal with a leg that has lost power (Fig. 3a:C3). Finally, the video illustrates how the Intelligent Trial and Error Algorithm can be applied to the second robot and to different damage conditions.

### Video S2

This video can be viewed at: <http://youtu.be/ycLspV5lXK8>

#### A Behavior-Performance Map Containing Many Different Types of Walking Gaits.

In the behavior-performance map 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.

## Supplementary References

30. A. E. Eiben and J. E. Smith. *Introduction to evolutionary computing*. Springer, 2003.
31. J.-B. Mouret and J. Clune. "Illuminating search spaces by mapping elites". In: *arXiv preprint arXiv:1504.04909* (2015).
32. D. J. Lizotte, T. Wang, M. H. Bowling, and D. Schuurmans. "Automatic Gait Optimization with Gaussian Process Regression." In: *Proceedings of the the International Joint Conference on Artificial Intelligence (IJCAI)*. Vol. 7. 2007, pp. 944–949.
33. E. Brochu, V. M. Cora, and N. De Freitas. "A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning". In: *arXiv preprint arXiv:1012.2599* (2010).
34. J. Snoek, H. Larochelle, and R. P. Adams. "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Advances in Neural Information Processing Systems 25 (NIPS)*. 2012, pp. 2951–2959.
35. T. L. Griffiths, C. Lucas, J. Williams, and M. L. Kalish. "Modeling human function learning with Gaussian processes". In: *Advances in Neural Information Processing Systems 21 (NIPS)*. 2009, pp. 553–560.
36. A. J. Booker, J. E. Dennis Jr, P. D. Frank, D. B. Serafini, V. Torczon, and M. W. Trosset. "A rigorous framework for optimization of expensive functions by surrogates". In: *Structural optimization* 17.1 (1999), pp. 1–13.
37. A. I. J. Forrester and A. J. Keane. "Recent advances in surrogate-based optimization". In: *Progress in Aerospace Sciences* 45.1 (2009), pp. 50–79.
38. Y. Jin. "Surrogate-assisted evolutionary computation: Recent advances and future challenges". In: *Swarm and Evolutionary Computation* 1.2 (2011), pp. 61–70.
39. T. W. Simpson, T. M. Mauery, J. J. Korte, and F. Mistree. "Comparison of response surface and kriging models for multidisciplinary design optimization". In: *American Institute of Aeronautics and Astronautics* 98.7 (1998), pp. 1–16.
40. D. R. Jones, M. Schonlau, and W. J. Welch. "Efficient global optimization of expensive black-box functions". In: *Journal of Global optimization* 13.4 (1998), pp. 455–492.
41. J. Sacks, W. J. Welch, T. J. Mitchell, H. P. Wynn, et al. "Design and analysis of computer experiments". In: *Statistical science* 4.4 (1989), pp. 409–423.
42. R. Calandra, A. Seyfarth, J. Peters, and M. P. Deisenroth. "An experimental comparison of Bayesian optimization for bipedal locomotion". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2014.
43. B. Matérn et al. "Spatial variation. Stochastic models and their application to some problems in forest surveys and other sampling investigations." In: *Meddelanden fran statens Skogsforskningsinstitut* 49.5 (1960).
44. M. L. Stein. *Interpolation of spatial data: some theory for kriging*. Springer, 1999.
45. A. V. Fiacco and G. P. McCormick. *Nonlinear programming: sequential unconstrained minimization techniques*. Vol. 4. Siam, 1990.
46. I. Dryanovski, R. G. Valenti, and J. Xiao. "Fast visual odometry and mapping from rgb-d data". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2013, pp. 2305–2310.
47. 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.
48. A. Sproewitz, R. Moeckel, J. Maye, and A. Ijspeert. "Learning to move in modular robots using central pattern generators and online optimization". In: *The International Journal of Robotics Research* 27.3-4 (2008), pp. 423–443.
49. 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". In: *Proceedings of ECAL* (2011), pp. 890–897.
50. J. Clune, K. Stanley, R. Pennock, and C. Ofria. "On the performance of indirect encoding across the continuum of regularity". In: *IEEE Transactions on Evolutionary Computation* 15.3 (2011), pp. 346–367.
51. J. Clune, B. Beckmann, C. Ofria, and R. Pennock. "Evolving coordinated quadruped gaits with the HyperNEAT generative encoding". In: *Proceedings of the IEEE Congress on Evolutionary Computation*. 2009, pp. 2764–2771.
52. S. Lee, J. Yosinski, K. Glette, H. Lipson, and J. Clune. "Evolving gaits for physical robots with the HyperNEAT generative encoding: the benefits of simulation." In: *Applications of Evolutionary Computing*. Springer, 2013.
53. D. Wilson. "Insect walking". In: *Annual Review of Entomology* 11.1 (1966), pp. 103–122.
54. U. Saranli, M. Buehler, and D. Koditschek. "Rhex: A simple and highly mobile hexapod robot". In: *The International Journal of Robotics Research* 20.7 (2001), pp. 616–631.
55. J. Schmitz, J. Dean, T. Kindermann, M. Schumm, and H. Cruse. "A biologically inspired controller for hexapod walking: simple solutions by exploiting physical properties". In: *The biological bulletin* 200.2 (2001), pp. 195–200.
56. X. Ding, Z. Wang, A. Rovetta, and J. Zhu. "Locomotion analysis of hexapod robot". In: *Proceedings of Conference on Climbing and Walking Robots (CLAWAR)* (2010), pp. 291–310.
57. S. Steingrube, M. Timme, F. Wörgötter, and P. Manoonpong. "Self-organized adaptation of a simple neural circuit enables complex robot behaviour". In: *Nature Physics* 6.3 (2010), pp. 224–230.
58. F. Delcomyn. "The Locomotion of the Cockroach *Pariplaneta americana*". In: *Journal of Experimental Biology* 54.2 (1971), pp. 443–452.
59. S. Thrun, W. Burgard, D. Fox, et al. *Probabilistic robotics*. MIT press Cambridge, 2005.
60. M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba. "A solution to the simultaneous localization and map building (SLAM) problem". In: *IEEE Transactions on Robotics and Automation* 17.3 (2001), pp. 229–241.
61. M. Tesch, J. Schneider, and H. Choset. "Using response surfaces and expected improvement to optimize snake robot gait parameters". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2011, pp. 1069–1074.
62. 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)*. Vol. 3. IEEE, 2004, pp. 2619–2624.
63. M. S. Erden and K. Leblebicioğlu. "Free gait generation with reinforcement learning for a six-legged robot". In: *Robotics and Autonomous Systems* 56.3 (2008), pp. 199–212.
64. D. J. Christensen, J. C. Larsen, and K. Stoy. "Fault-tolerant gait learning and morphology optimization of a polymorphic walking robot". In: *Evolving Systems* 5 (2014), pp. 21–32.
65. S. Mahdavi and P. Bentley. "Innately adaptive robotics through embodied evolution". In: *Autonomous Robots* 20.2 (2006), pp. 149–163.
66. S. Koos, A. Cully, and J.-B. Mouret. "Fast damage recovery in robotics with the T-resilience algorithm". In: *The International Journal of Robotics Research* 32.14 (2013), pp. 1700–1723.
67. G. Hornby, S. Takamura, T. Yamamoto, and M. Fujita. "Autonomous evolution of dynamic gaits with two quadruped robots". In: *IEEE Transactions on Robotics* 21.3 (2005), pp. 402–410.
68. T. Barfoot, E. Earon, and G. D'Eleuterio. "Experiments in learning distributed control for a hexapod robot". In: *Robotics and Autonomous Systems* 54.10 (2006), pp. 864–872.
69. S. Koos, J.-B. Mouret, and S. Doncieux. "The transferability approach: Crossing the reality gap in evolutionary robotics". In: *IEEE Transactions on Evolutionary Computation* 17.1 (2013), pp. 122–145.