Automated Robot Function Recovery after Unanticipated Failure or Environmental Change using a Minimum of Hardware Trials

Josh C. Bongard Hod Lipson
Sibley School of Mechanical and Aerospace Engineering
Cornell University, Ithaca, New York 14850
Email: [JB382|HL274]@cornell.edu Phone: (607) 255-0396

Abstract

Recovering functionality after unanticipated damage or environmental change, using a minimum amount of hardware testing, is a desirable and under-explored topic in evolutionary hardware and evolutionary robotics. In a previous paper we introduced a two-stage evolutionary algorithm, which we call the estimation-exploration algorithm, that evolves a robot simulator to accurately describe what damage a 'physical'[1] robot has undergone, and then evolves a compensatory neural network in the evolved simulator that, when downloaded to the 'physical' robot, restores functionality. Here we introduce a new fitness metric that allows the algorithm to correctly describe not only complete but also partial failures, and also allows the algorithm to disambiguate between internal damage and external environmental change, based solely on sensory feedback. In most cases only four hardware evaluations are necessary in order to restore complete functionality to the 'physical' robot.

1 Introduction

For a robot to function for long periods of time in a hostile, unknown or remote environment, it must be able to deal autonomously with uncertainty: specifically, unanticipated internal damage or external environmental change. The recent difficulties with JPL's two Mars rovers provide a dramatic example: both robots suffered different, unanticipated partial failures [19]. Automatic recovery is most acute in such instances where human operators cannot manually repair or provide compensation for failure. In this work we are concerned with catastrophic, highly nonlinear robot faults that require recovery controllers qualitatively different from the original controller.

Some work has been done on employing evolutionary algorithms to restore functionality after some unanticipated damage has occurred, but all of this work relies on massive numbers of hardware trials: robot recovery has been demonstrated in [18] and [3], and for electronic circuits in [7] and [15]. However, repeated generate-and-test algorithms for robotics is not desirable for several reasons: repeated trials may exacerbate damage and drain limited energy; long periods of time are required for repeated hardware trials; damage may require rapid compensation (eg., power drain due to coverage of solar panels); and repeated trials continuously change the state of the robot, making damage diagnosis difficult.

Several types of plastic neural network controllers have been proposed that allow for rapid, lifetime adaptation to external perturbation ([10], [9] and [12]). Furthermore, Keymeulen *et al.* [14] have formulated an algorithm that continuously updates an internal model of sensor input/world response data obtained from a wheeled robot, and uses this model to evolve and download controllers to the robot during task execution. They have demonstrated that their algorithm greatly reduces the number of required hardware trials, compared to a similar model-free algorithm. However none of these approaches generate a hypothesis describing what particular change the robot or its environment has experienced. Secondly, the evolved controllers have not been shown capable of fundamental reorganization when faced with unanticipated, catastrophic failure (such as the separation of a limb), as is demonstrated here for our proposed algorithm.

Here we describe our estimation-exploration algorithm, which, for the results presented here, requires only four hardware trials (on average) to restore complete functionality. The algorithm automatically evolves two separate structures: a robot simulator that explains unanticipated internal damage or external environmental change suffered by the real robot; and a compensatory neural network controller that restores functionality to the real robot, given the evolved robot simulator.

Srinivas [22] was one of the first researchers to study error diagnosis and recovery, but his approach, along with subsequent approaches ([8, 1, 11, 13, 23]), required online operation (repeated testing on the physical robot), and could not handle unanticipated errors. Baydar and Saitou [3] proposed the first offline error diagnostic and recovery system, which relies on Bayesian inference for error diagnosis, and

[1] Currently, the 'physical' robot is not physical but simulated. The algorithm is currently being applied to a physical robot.

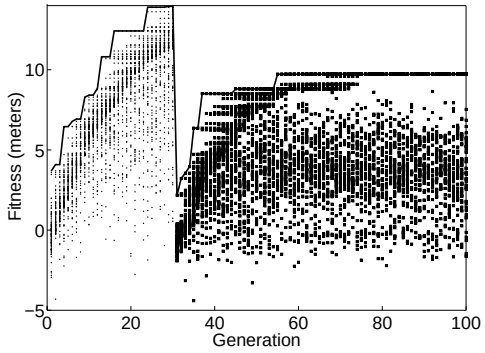

Figure 1. Evolutionary progress for robot damage recovery. Each dot represents a simulated robot evaluation. Each square represents a 'hardware' evaluation.

Genetic Programming [16] for error recovery. However their algorithm also only handles pre-specified error types.

Mahdavi and Bentley [18] recently demonstrated an online evolutionary algorithm that automatically recovers behavior for a physical robot. However after damage the physical robot required 400 hardware trials and nearly seven hours to recover 72% of its original functionality. Figure 1 shows the progress of a similar evolutionary algorithm applied to a simulated quadrupedal robot (see Figure 3) in which the first part of evolution is performed in a robot simulator; the best controller is downloaded to a 'physical' robot (another simulated robot); the lower part of one of the robot's legs is broken off; and compensatory neural controllers are then evolved on the cripped 'physical' robot. This run required a total of 3550 'hardware' trials and provided only 80% recovery (details regarding the robot and algorithm are provided in later sections).

Due to the recent advances in simulation it has become possible to automatically evolve both the controller and morphology of simulated robots [21, 17, 2, 4]. Here we also use evolutionary algorithms to co-evolve bodies and brains, but employ an inverse process: in addition to an exploration phase that evolves a robot controller given a fixed robot simulator, an estimation phase evolves a robot simulator (including the simulated robot's morphology and its simulated environment) based on feedback from the real robot, after it has used the evolved controller.

In a previous paper [6] we have shown that such a coevolutionary algorithm can successfully diagnose several types of discrete failures, such as the separation of a body part or the complete failure of a sensor or motor (or a combination of such failures) using at most four 'hardware' evaluations. This stands in contrast to all other approaches to automated recovery so far, which can only compensate for one or a few pre-specified failure types, and do not provide a diagnosis of the failure: our approach provides an evolved robot simulator that explains the failure.

In this paper we describe a new metric that quantitatively compares behaviors between robots that have differing morphologies and/or exist in differing environments. This metric then serves as a fitness measure when comparing the behavior of the 'physical' robot against simulated robots during the estimation phase: robot simulators that better approximate the behavior of the real robot better explain the real robot's failure or novel environment. We show here how this metric allows for functionality recovery after complete or partial failure, or unanticipated environmental change.

The algorithm and the new metric are described in the next section. Section 3 presents results from the application of the algorithm to a simulated quadrupedal robot. Section 4 provides an explanation for why the new fitness metric allows for successful functionality recovery over a wider range of unanticipated situations, and the final section provides some concluding remarks and avenues of further study.

2 Methods

2.1 Algorithm Overview

The algorithm for automated recovery has two phases: controller evolution (the exploration phase) and simulator evolution (the estimation phase). The exploration phase evolves a controller for the 'physical' robot using a robot simulator. The estimation phase evolves a robot simulator, given sensor data generated by the 'physical' robot using the controller evolved in the previous phase.

Initially, the exploration phase is run, given an approximate simulation of the robot and its environment. When the estimation phase terminates, the best evolved controller is downloaded to the 'physical' robot. The robot then behaves, and the resulting sensor data is then supplied, along with the evolved controller, to the estimation phase. The estimation phase evolves the simulator so that the simulated robot, given the previously evolved controller, produces the same sensor data as the real robot. The new, better simulator is passed to the exploration phase, and the cycle repeats until functionality is restored to the real robot and an accurate simulator describing the real robot and its environment is produced. After each hardware trial, the estimation phase has a new data pair (controller/sensor data) to use for evolving the simulator; the more data the estimation phase obtains from the real robot, the better it is at evolving an accurate simulation.

Figure 2 outlines the flow of the algorithm, as well as a general outline of the algorithm applied to any hidden nonlinear system. In general, the exploration phase evolves some experiment to perform on the hidden target system, given a description of that system; the data returned by the hidden system is used to further refine that description. The cycle repeats until either a desirable behavior is achieved on the target system, or a sufficiently accurate description of the system is evolved. In this work, the controller serves

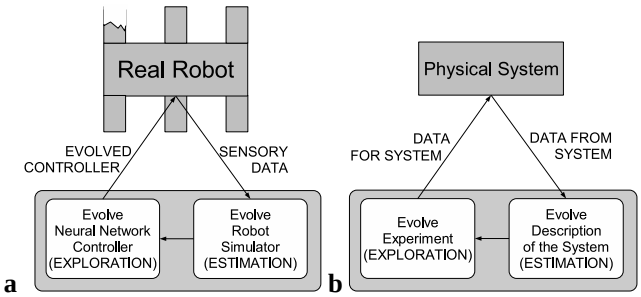

Figure 2. a: Automated function recovery for a damaged robot using the estimation-exploration algorithm. b: Automated inference of a hidden nonlinear system using the same algorithm.

as the experiment driving the target system, and the robot simulator serves as the description of the system.

2.2 Experimental Setup

The Robot. In this work a quadrupedal robot was simulated and used to test the algorithm. The simulator runs onboard a 'physical' robot, which in this case is also simulated. The robot simulator is based on Open Dynamics Engine, an open-source 3D dynamics simulation package [20]. The simulated robot is composed of nine three-dimensional objects, connected with eight one-degree of freedom rotational joints. The joints are motorized, and can rotate through $[-\pi/4, \pi/4]$ radians. The robot is shown in Figure 3. The robot also has four binary touch sensors (1 if the lower leg is in contact with the ground plane, and -1 otherwise), and four proprioceptive sensors that return values in $[-1, 1]$ commensurate with the angle of the joint to which they are attached.

While using the evolved controller produced by the first pass through the exploration phase, the 'physical' robot suffers some unanticipated failure or enters a novel environment. The robot is then stopped, and the resultant sensor logs are transferred back to the estimation phase. During subsequent 'hardware' trials, the damaged robot attempts to move using the compensatory controllers evolved by subsequent passes through the exploration phase.

The Controllers. The robots are controlled by a neural network, which receives sensor data from the robot at the beginning of each time step of the simulation into its input layer, propagates those signals to a hidden layer containing three hidden neurons, and finally propagates the signals to an output layer. The neural network architecture is shown in Figure 4. Two types of sensors are used: touch sensors and angle sensors: the touch sensors are binary, and indicate whether the object containing them is in contact with the ground plane or not; the angle sensors return a value commensurate with the flex or extension of the joint to which they are attached. Neuron values and synaptic weights are scaled or lie in the range $[-1.00, 1.00]$. A thresholding activation function is applied at the neurons.

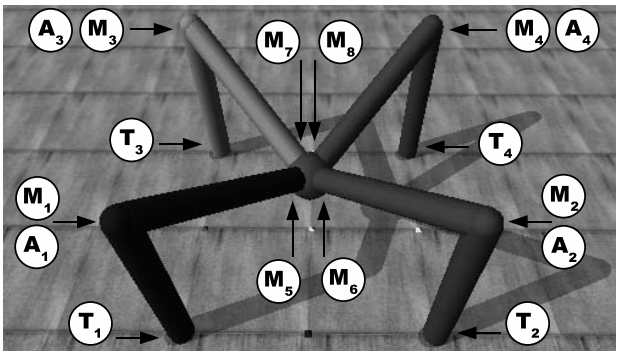

Figure 3. The quadrupedal robot. T_i indicates touch sensors; A_i indicates angle sensors; M_i indicates motorized joints.

There is one output neuron for each of the motors actuating the robot: the values arriving at the output neurons are scaled to desired angles for the joint corresponding to that motor. For both robots here, joints can flex or extend to $\frac{\pi}{4}$ away from their default starting orientation. The angles are translated into torques using a PID controller, and the simulated motors then apply the resultant torque. The physical simulator then updates the positions, orientations and velocities of the robot's body parts by adding these torques to the external forces (gravity, friction, momentum and collision with the ground plane) acting on the objects. The resulting motions cause the robot to move, and the cycle is repeated for a set number of time steps.

The Exploration Phase. The genomes of the exploration phase's evolutionary algorithm are strings of floating-point values, which encode the synaptic weights of the neural network controller. There are a total of 68 synapses, giving a genome length of 68 values. The encoded synaptic weights are represented to two decimal places, and lie in the range [-1.00,1.00].

At the beginning of each run a random population of 200 genomes is generated. If there are any stored controllers evolved by previous passes through the exploration phase, these are downloaded into the population. The robot is then evaluated in the simulator for 1000 time steps, and the fitness of that controller is computed. The fitness of a controller is how far, in meters, it causes the robot to move forward during this time period. Once all of the genomes in the population have been evaluated, they are sorted in order of decreasing fitness, and the 100 least fit genomes are deleted from the population. One hundred new genomes are selected to replace them from the remaining 100, using tournament selection, with a tournament size of 3. Each floating-point value of a copied genome has a 1 per cent chance of undergoing a point mutation (replacement of the evolved value with a new random value). Of the 100 copied and mutated genomes, 24 pairs are randomly selected and undergo one-point crossover.

The Estimation Phase. The estimation phase evolves a robot simulator that describes the failure incurred by the

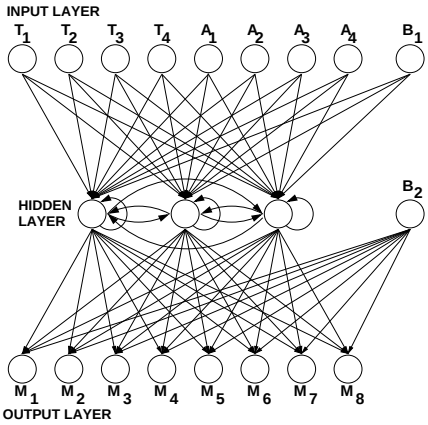

Figure 4. The neural network used to control the robot shown in Figure 3. T_i indicates touch sensor neurons; A_i indicates angle sensor neurons; B_i indicates bias neurons; and M_i indicates motor neurons.

'physical' robot, or a description of its new environment. Using only sensory data arriving from the 'physical' robot, this phase can distinguish between these two possibilities automatically, as shown in the next section.

For each genome in the estimation phase, the simulated robot is broken or the simulated environment is modified according to the genome's encoded instructions. The robot is then evaluated using each of the best controllers evolved by the previous passes through the exploration phase. During the first pass through the estimation phase, there is only a single controller available.

In a previous version of the algorithm ([6]), the function for determining the fitness of the simulator described by genome g_j was set to

$$f(g_j) = \frac{\sum_{i=1}^{c} |f_{\text{phy}}(i) - f_{\text{sim}}(i)|}{c}, \quad (1)$$

where c is the number of controllers evolved so far by the exploration phase, $f_{\text{phy}}(i)$ is the forward distance traveled by the 'physical' robot using evolved controller i, and $f_{\text{sim}}(i)$ is the forward distance traveled by the simulated robot using evolved controller i after the robot simulator has been modified by the instructions encoded in g_j. The fitness function was therefore an attempt to minimize the difference between the simulated and 'physical' robots' fitness values.

However the observation was made that in many cases the simulated robot would exhibit wildly different behaviors even when it very closely approximated the damaged 'physical' robot. This result is not surprising due to the fact that the robot is a highly coupled, non-linear system: thus similar initial conditions (two identical robots with identical controllers but only similar damages) are expected to rapidly diverge in behavior over time.

This point is illustrated by Figure 5a, which reports the values returned by sensor A_1 during three separate evaluations of the simulated robot: in each evaluation the simulated robot uses the same controller, but each robot suffers three different failures. As can be seen, the time series of the two related failures stay correlated for a long period of time (divergence occurs around $t = 250$), while the unrelated damaged robot diverges from the first two related robots earlier ($t = 50$). Later, all three of the time series become uncorrelated due to the coupled non-linear nature of the system. Based on this observation a new fitness metric for the estimation phase was formulated, in which the fitness of genome p is given by:

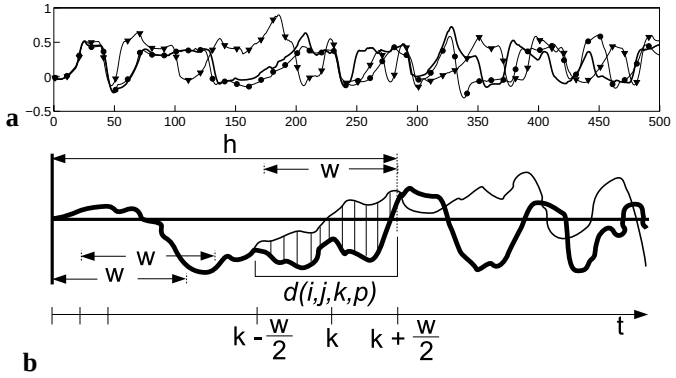

Figure 5. a: Each line represents the values of sensor A_1 when the simulated robot shown in Figure 3 undergoes three separate failures. The thick line indicates the sensor values when the one of the robot's motors weakens by 10%; the line with circle markers when the same motor has been weakened by 20%; and the line with triangle markers when one of the touch sensors has failed by 50%. Each of the three robots is evaluated using the same evolved controller. Only the first 500 time steps of the 1000 time step evaluation period are shown. b: Outline of the rolling mean fitness metric.

$$f(g_p) = \frac{\sum_{i=1}^{c} \sum_{j=1}^{n} \sum_{k=w/2}^{h-w/2} d(i,j,k,p)}{cn(h-w)}, \quad (2)$$

$$d(i,j,k,p) = \frac{\sum_{t=k-w/2}^{t=k+w/2} |s_{\text{phy}}^{(i,j)}(t) - s_{\text{sim}}^{(i,j,p)}(t)|}{w}. \quad (3)$$

In this formulation i is the number of controllers evolved for the 'physical' robot so far; j is the number of sensors contained in the robot (here $j = 8$); h is some header length, which indicates how much of the initial time series data to use (for the experiments reported here, $h = 20$; and w indicates the width of a time window within this time header (here $w = 10$). $d(i,j,k,p)$ represents the average difference between the sensor values of the 'physical' robot and the sensor values of the simulated robot when the simulator is modified by the instructions encoded in genome p, over

a short time period $[k - w/2, k + w/2]$. $s_{\text{phy}}^{(i,j)}(t)$ indicates the activation of the jth sensor at time t obtained from the 'physical' robot when it is using controller i, and $s_{\text{sim}}^{(i,j,p)}(t)$ indicates the activation of the jth sensor at time t from the simulated robot using controller i in the simulation modified by genome p. We refer to this metric as the *rolling mean* metric, as it aims to compare sets of average sensor activations over a short initial time period. Figure 5b depicts this metric graphically. During subsequent passes through this stage, there are additional pairs of evolved controllers and sets of sensor data returned by the 'physical' robot when using those controllers, respectively ($i > 1$).

The genomes of the estimation phase, like the exploration phase, are strings of floating-point values. Each genome in the estimation phase is composed of a set of four genes: each gene represents a modification to be made to the robot simulator. There are six different possible modifications: a body part breaks off (the joint separates); a body part increases in mass by some amount (from 100% to 200%); a motor weakens by some percentage (0% to 100%); a sensor fails by some percentage (0% to 100%); a joint jams by some percentage (0% to 100%); or the floor is canted horizontally (-30 to 30 degrees).

Each of the four genes encoded in a genome is comprised of four floating-point values, giving a total genome length of 16. Like the repair EA, each of the values is represented to two decimal places, and lies in $[-1.00, 1.00]$.

The first floating-point value of a gene is rounded to an integer in $[0, 1]$ and denotes whether the gene is dormant or active: ie., whether the situation encoded by that genome is applied to the robot model or not. If the gene is active, the second floating-point value is rounded to an integer in $[0, 5]$, and indicates which of the six situations should be applied to the simulation. Depending on which situation is encoded by the second value, the third value is scaled to: $[0, j - 1]$ if the damage applies to a joint; $[0, m - 1]$ if the damage applies to a motor; $[0, s - 1]$ if the damage applies to a sensor; $[0, b - 1]$ if the damage applies to a body part; or is disregarded if the situation implies an environmental change. The fourth value is then: treated as a percentage if the situation is a partial failure; disregarded if the failure is complete (joint separation); or scaled to a value in $[-30.0, 30.0]$ if the situation is a horizontally canted floor. This encoding gives a search space of $200^{16} = 6.5 \times 10^{36}$ different genomes. As the values are discretized as explained above, there are at most 4 genes $\times$ 2 active/inactive gene $\times$ 6 situations $\times$ 9 body parts $\times$ 200 percentages $= 86400$ sets of simulator modifications, although some of these are not unique because ordering is irrelevant, and modifications encoded by inactive genes are not applied.

At the termination of the estimation phase the best evolved simulator modification is passed to the exploration phase, and the cycle continues. This modification, along with modifications evolved in previous passes through the estimation phase, are used to seed the initial random population of genomes during the next pass through the estimation phase.

Table 1. Unanticipated Situations Tested

Case	Explanation
1	One motor weakens by 50%.
2	One body part increases in mass by 200%.
3	One the entire legs breaks off.
4	One of the entire legs breaks off, and a sensor fails by 50%.
5	An angle sensor fails by 50%.
6	One of the joints jams by 50%.
7	One of the entire legs breaks off, and one of the joints jams by 50%.
8	One of the entire legs breaks off, one of the joints jams by 50%, and one of the sensors breaks by 50%.
9	Nothing breaks.
10	The robot stands on a 30 degree horizontal slope.
11	One of the hidden neurons fails by 50%.
12	Two motor neurons output the same value.
13	A body part decreases in mass by 50%.

3 Results

The robot was exposed to 13 different unanticipated situations, which are listed in Table 1. The algorithm described above was run 10 times on each situation. The estimation phase attempts to evolve a simulator modification that will describe this situation: for each of the 13 situations, there are about 5 or 6 different genomes that will describe it perfectly, out of the 86400 possible situations.

For each run of the algorithm described, the exploration phase was run once to generate the initial evolved controller, and then the 'physical' robot was exposed to one of the 13 unanticipated situations listed in Table 1. Sensor data is then returned from the 'physical' robot to the algorithm. The loop shown in Figure 2a is then traversed four times, giving a total of five passes through the exploration phase, four hardware trials and four passes through the estimation phase. Each phase is run for 40 generations, using a population size of 200 genomes (this produces a total of $40 \times 9 = 360$ generations for the algorithm). A total of 130 independent runs of the algorithm were performed (10 independent runs for each of the 13 situations). Each run requires about 4 hours of computation on a 1GHz PC.

An additional 130 independent runs were performed as a control: the control algorithm is identical to the proposed algorithm, but rather than using the rolling mean metric described in Equation 3 to evaluate the fitness of a simulator modification, the original metric described by Equation 1 was used. Figure 6a shows a typical run of the control algorithm when one of the angle sensors fails (situation 5). Figure 6b shows a typical run using the proposed algorithm

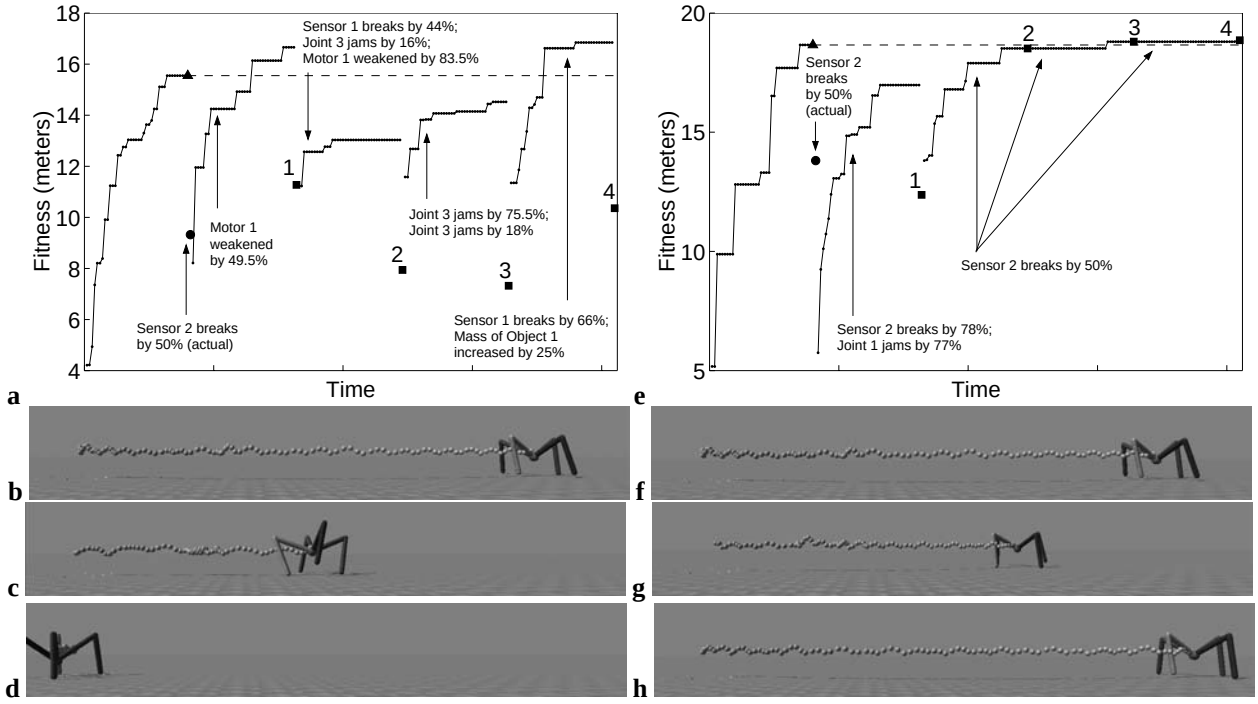

Figure 6. a: The evolutionary progress of the control algorithm on situation 5: one of the angle sensors breaks by 50%. The dotted lines indicate the progress of the five passes of the exploration phase. The captions indicate the best simulator modifications evolved by the four passes through the estimation phase. The triangle shows the original fitness (distance traveled in meters) of the 'physical' robot. The circle indicates distance traveled after suffering the failure. The squares indicate the distance traveled by the damaged 'physical' robot during each of the four hardware trials. b-d: Trajectories of the 'physical' robot's motion: before the unanticipated situation (b); after encountering the unanticipated situation (c); and during the fourth hardware trial. e-h: The same data for a typical run of the proposed algorithm for the same situation, but using the rolling mean metric.

for the same situation. Figure 7 compares the average performance of the control and proposed algorithms for all 13 unanticipated situations.

4 Discussion

Figure 6 shows that the control algorithm, in that instance, was unable to discover the correct simulator modification that would explain the unanticipated situation the 'physical' robot had encountered. The proposed algorithm, on the other hand, closely approximated the actual situation during the first pass through the estimation phase: it guessed correctly which sensor had failed, but was off in terms of the magnitude of failure (78% compared to 50%), and also incorrectly surmised that a joint had become severely jammed. The second pass through the estimation phase, however, aided by the second set of sensor data returned by the 'physical' robot, was able to generate the correct modification. The two final passes retained the correct modification, allowing the exploration phase to evolve a controller which actually outperformed the original controller.

Figure 7 makes clear that the proposed algorithm significantly outperforms the control algorithm. Specifically, the control algorithm was unable to recover functionality for unanticipated situations 1, 2, 4, 5 and 6, as evidenced by the statistical insignificance between the average distance traveled by the 'physical' robot after encountering the situation, and during the fourth hardware trial. However the proposed algorithm was able to restore functionality for these five situations successfully. It is notable that these situations involve a partial failure; in other words the failure is described by a percentage.

This seems to suggest that when the search space of the estimation phase is small (one of eight possible joint breakages is correct), such as the situation when a leg breaks off, the estimation phase can find the correct modification through random search. In order to confirm this explanation, all 130 evolutionary histories generated by the first pass through the estimation for the control algorithm are plotted in Figure 8a, and the 130 evolutionary histories of the first pass through the estimation phase for the proposed algorithm are plotted in 8b.

In order to quantify evolutionary activity in each of these

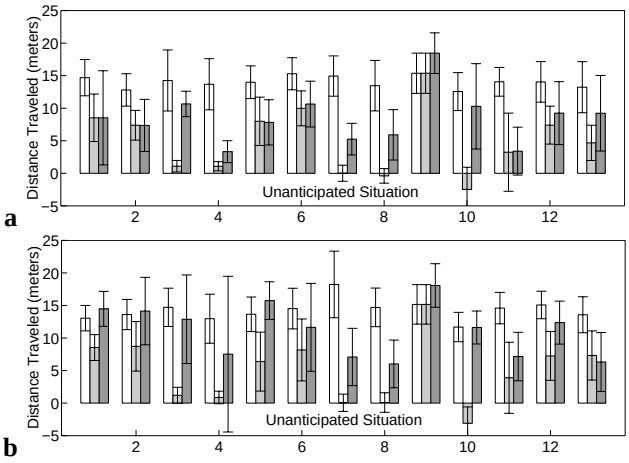

Figure 7. a: The average performance of the control algorithm for all unanticipated situations. b: The average performance of the actual algorithm for all unanticipated situations. The white bars indicate the average distance traveled by the 'physical' robot before encountering the unanticipated situation; the light gray bars after it has encountered the situation; and the dark gray bars indicate the distance traveled by the 'physical' robot during the fourth hardware trial.

passes, each generation was checked to see whether a dominant had appeared in the population. A *dominant* is defined as a new genome that has a higher fitness than all of the genomes from the previous generation. Because our fitness evaluation is free of noise, this implies that the appearance of a dominant in the population indicates the discovery of a new best solution to the problem, and a subsequent movement of the population to a higher region of the fitness landscape. A population that produces many dominants over time indicates the population is traversing a landscape with more gradients than one that produces fewer dominants.

Figure 8 makes clear that the estimation phase of the proposed algorithm produces many dominants, indicating that there is much evolutionary activity: several passes through the estimation phase produce 19 dominants (Figure 8b). Conversely the estimation phase of the control algorithm displays little evolutionary activity: the best passes only produce four dominants. Because the only difference between the estimation phases in the two algorithms is the fitness function, we can conclude that the new fitness metric introduces many gradients into the fitness landscape, allowing the proposed algorithm to discover the correct explanation of the unanticipated situation more often.

In the ninth unanticipated situation, no damage or environmental change occurred: in this case the algorithm usually guesses correctly that nothing has changed. This has the effect of continuing to evolve the performance of the controller originally evolved by the first pass through the

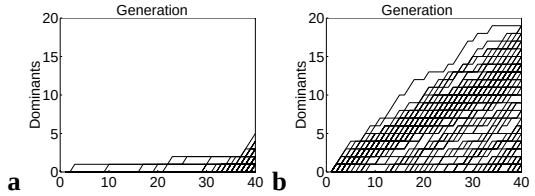

Figure 8. Evolutionary activity in the control and proposed algorithms. Each trajectory corresponds to a single pass through the estimation phase of either the control (a) or the proposed (b) algorithm. There are a total of $4 \times 10 \times 13 = 520$ overlapping trajectories for each algorithm.

exploration phase: this explains the performance increase observed in the fourth hardware trial, compared to original performance of the 'physical' robot.

Note that both algorithms were able to successfully recover when faced with an unanticipated environmental change: the horizontal canting of the ground plane (situation 10 in Table 1). This was probably due to the drastic effect on all sensors induced by this change. It is important to note that the proposed algorithm can distinguish between internal damage and external environmental change based solely on indirect information: distance traveled (in the control algorithm) or sensory data (in the actual algorithm).

Finally, the last three unanticipated situations are truly unanticipated: the estimation phase can only modify the simulate to approximate the state of the 'physical' robot and its environment. Not surprisingly, both the control and proposed algorithm have difficulty evolving a compensatory controller for these situations. However, the proposed algorithm does seem to do better at recovering from the 11th and 12th situations, although the average recovery is not statistically significant. Further study is required in order to understand how truly unanticipated situations can best be approximated by a set of simulator modifications.

5 Conclusions

In order for robots—and hardware systems in general—to survive for long periods of time in remote and uncertain environments, they must carry algorithms that allow them to autonomously adapt to a wide range of unanticipated situations, without requiring extensive hardware testing. In this paper we have described a two-stage algorithm that can automatically diagnose and recover from a wide range of unanticipated internal damage or environmental change using only four hardware trials. This stands in sharp contrast to all other past attempts at evolving robustness or error recovery for evolutionary robotics and evolvable hardware, which require many evaluations to be performed on the actual system, and/or do not produce a diagnosis of the unanticipated change.

Here we have introduced a new fitness metric which

better allows the proposed algorithm to evolve a simulator that describes the unanticipated situation encountered by the robot. This metric has been shown to allow the algorithm to autonomously distinguish between internal damage, environmental change, or no change, based solely on data returned by the robot's sensors: in other words, no additional sensors are required for error diagnosis.

The true power of this algorithm, however, lies in its generality: we hold that our algorithm can be applied to most coupled, non-linear systems, and have so far have also applied the algorithm, with little modification, to the problem of gene network inference [5]. Future avenues of study will include: replacing the simulated 'physical' robot with an actual physical robot; investigating how best to describe an unanticipated situation that cannot be explained perfectly with any automated combination of simulator modifications; and generalizing the algorithm to more coupled, non-linear systems.

Acknowledgements

This research was conducted using the resources of the Cornell Theory Center, which receives funding from Cornell University, New York State, federal agencies, foundations, and corporate partners. This work was supported by the U.S. Department of Energy, grant DE-FG02-01ER45902.

References

[1] M.G. Abu-Hamdan and A.S. El-Gizawy. Computer aided monitoring system for flexible assembly operations. *Computers in Industry*, 34:1–10, 1997.

[2] A. Adamatzky, M. Komosinski and S. Ulatowski, S. Software review: Framsticks. In *Kybernetes: The International Journal of Systems & Cybernetics*, 29:1344–1351, 2000.

[3] C.M. Baydar and K. Saitou. Off-line error prediction, diagnosis and recovery using virtual assembly systems. In *IEEE Intl. Conf. on Robotics and Automation*, pp. 818–823, 2001.

[4] J.C. Bongard. Evolving modular genetic regulatory networks. In *Proceedings of The IEEE 2002 Congress on Evolutionary Computation (CEC2002)*, pp. 1872–1877, 2002.

[5] J.C. Bongard and H. Lipson. Automating genetic network inference with minimal physical experimentation using coevolution. To appear in *Proceedings of the 2004 Genetic and Evolutionary Computation Conference (GECCO)*, Seattle, WA.

[6] J.C. Bongard and H. Lipson. Automated damage diagnosis and recovery for remote robotics. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2004.

[7] D.W. Bradley and A.M. Tyrrell. Immunotronics: novel finite-state-machine architectures with built-in self-test Using self-nonself differentiation. In *IEEE Transactions on Evolutionary Computation*, 6(3):227–38, 2002.

[8] S. Brnyjolfsson and A. Arnstrom. Error detection and recovery in flexible assembly systems. In *Intl. Journal of Advanced Mfg. Technology*, 5:112–125, 1997.

[9] E.A. Di Paolo. Homeostatic adaptation to inversion of the visual field and other sensorimotor disruptions. In *From Animals to Animats 6*, pp. 440–449, MIT Press, Cambridge, MA, 2000.

[10] P. Eggenberger, A. Ishiguro, S. Tokura, T. Kondo and Y. Uchikawa. Toward seamless transfer from simulated to real worlds: A dynamically-rearranging neural network approach. In *European Workshop on Learning Robots (EWLR 8)*, pp. 44–60, Springer, Berlin, 1999.

[11] E.Z. Evans and S.G. Lee. Automatic generation of error recovery knowledge through learned activity. In *IEEE Intl. Conf. on Robotics and Automation*, 4:2915–2920, 1994.

[12] D. Floreano and J. Urzelai. Evolution of plastic control networks. In *Autonomous Robots*, 11(3):311–317, 2001.

[13] J.F. Kao. Optimal recovery strategies for manufacturing systems. In *European Journal of Operations Research*, 80:252–263, 1995.

[14] D. Keymeulen, M. Iwata, Y. Kuniyoshi and T. Higuchi. Online evolution for a self-adapting robotics navigation system using evolvable hardware. In *Artificial Life*, 4:359–393, 1998.

[15] D. Keymeulen, A. Stoica and R. Zebulum. Fault-tolerant evolvable hardware using field programmable transistor arrays. In *IEEE Transactions on Reliability, Special Issue on Fault-Tolerant VLSI Systems* 3(49):305–316, 2000.

[16] J.R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*, MIT Press. Cambridge, MA, 1992.

[17] H. Lipson and J.B. Pollack. Automatic design and manufacture of artificial lifeforms. In *Nature*, 406:974–978, 2000.

[18] S.H. Mahdavi and P.J. Bentley. An evolutionary approach to damage recovery of robot motion with muscles. In *Seventh European Conference on Artificial Life (ECAL03)*, pp. 248-255, Springer, Berlin, 2003.

[19] JPL Mars Exploration Rovers (2004) `http://www.jpl.nasa.gov/mer2004/`.

[20] `opende.sourceforge.net`

[21] K. Sims. Evolving 3D morphology and behaviour by competition. In *Artificial Life IV*, pp. 28–39, 1994.

[22] S. Srinivas. *Error Recovery in Robot Systems*. Ph.D. thesis, California Institute of Technology, 1977.

[23] M.L. Visinsky, J.R. Cavallaro and I.D. Walker. Expert system framework for fault detection and fault tolerance in robotics. In *Computers in Electrical Engineering*, (20)5:421–435, 1994.