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 that allows for the evolution of hypotheses regarding what damage a 'physical' robot (which at the moment is also a simulated robot) has undergone, and then evolution of a compensatory neural network to restore functionality, through the use of an onboard robot simulator. Sensory feedback from the 'physical' robot accelerates further evolution of hypotheses and compensatory controllers. Here we introduce a new fitness metric that allows the algorithm to correctly guess 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.

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 unanticipated external environmental change. The recent difficulties with JPL's Spirit and Opportunity Mars rovers is a dramatic example; both robots suffered different, unanticipated partial failures [14]. Automatic recovery is most acute in such instances, where human operators cannot manually repair or provide compensation for damage or failure. In this work we are concerned with catastrophic faults (highly nonlinear) that require recovery controllers that are 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 [13] and [3], and for electronic circuits in [6] and [10]. 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 changes the state of the robot, making damage diagnosis difficult.

Here we discuss our offline damage diagnosis and repair algorithm, which, for the results presented here, requires only four hardware trials to restore (on average) complete functionality. The algorithm allows the robot to automatically determine whether it has suffered some unanticipated internal damage or external environmental change, or whether no change has occurred (evolution of damage hypothesis), and then to evolve a compensatory neural network controller (evolution of functionality).

Srinivas [17] was one of the first researchers to study error diagnosis and recovery, but his approach, along with subsequent approaches ([7, 1, 8, 9, 18]), 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 Genetic Programming [11] for error recovery. However their algorithm also only handles pre-specified error types. The algorithm proposed here is demonstrated to automatically provide an approximate diagnosis and successfully recover from an unanticipated failure type.

Mahdavi and Bentley [13] recently demonstrated the ability of an online evolutionary algorithm to automatically recover behaviour 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 in which all compensatory neural controllers are evolved on the 'physical' robot after damage, leading to a

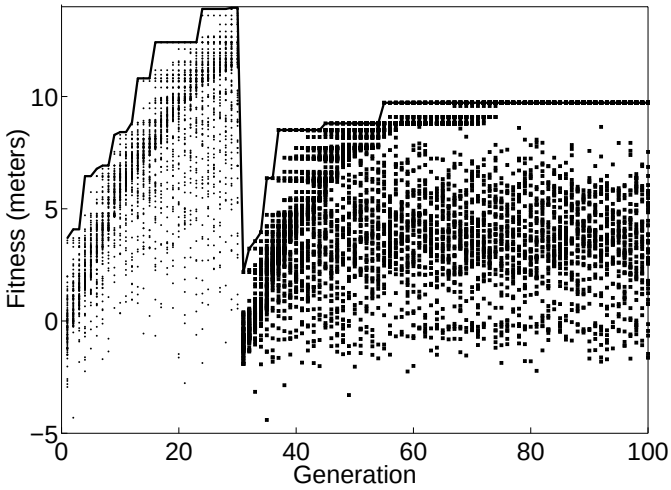

Figure 1. The evolutionary history for an evolving robot population. Controllers for forward locomotion are evolved on the simulated quadrupedal robot (Figure 3) for the first 30 generations (each dot represents one or more simulated robot evaluations). The controller is transferred to the 'physical' robot (also simulated), which undergoes damage case 1: the breakage of one of its lower legs. Evolution is continued on the physical robot (each square represents one or more physical robot evalutions). A total of 3550 evaluations only restore about 70% functionality.

total of 3550 hardware trials and only 70% 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 controller and morphological changes together for simulated robots, and measure behavioural effect [16, 12, 2, 4]. Here we also use evolutionary algorithms to co-evolve bodies and brains, but use an inverse process: in addition to a first stage that evolves a controller given a fixed morphology, a second stage evolves a morphology given the previously evolved controller; and instead of maximizing some behaviour (such as forward locomotion), we minimize some error (sensors values obtained from the physical robot, compared against sensor values obtained using the hypothesized morphology).

In a previous paper [5] we have shown that such a two-stage 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, using large number of hardware trials.

In this paper we describe a new fitness metric for the stage of the algorithm that evolves morphological hypotheses, and we show how it 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 stages: controller evolution and model hypothesis evolution. The algorithm also maintains a database, which stores pairs of data: an evolved controller and the sensor logs produced by the physical robot when that controller is used. The algorithm is composed of two separate evolutionary algorithms: the repair EA and the diagnostic EA. The repair EA generates controllers for the physical robot using a robot simulator. The diagnostic EA uses the simulator also, but evolves hypotheses regarding what failure or environmental change the physical robot has experienced. Evolved controllers are uploaded from the simulation to the physical robot, and sensor data are downloaded from the physical robot to the simulation. Figure 2 outlines the flow of the algorithm, along with a comparison against an algorithm for evolving function recovery all on a physical robot.

Stage I: Controller Evolution. The **repair EA** is used to evolve a controller for the simulated robot, such that it is able to perform some task. The first pass through this stage generates the controller for the intact physical robot: subsequent passes attempt to evolve a compensatory controller for the physical robot using the current best hypothesis generated by Stage II describing what damage or environmental change has occurred. When the repair EA terminates, the best controller from the run is transferred to and used by the physical robot.

Physical Robot Failure. 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 [15]. 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, and four proprioceptive sensors that return values in $[-1, 1]$ commensurate with the angle of the joint to which they are attached.

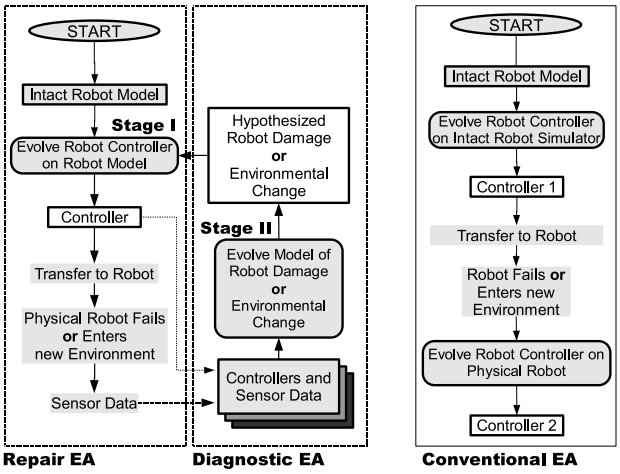

Figure 2. The two left-hand panels show the flow for the recovery algorithm proposed in this paper. The dotted arrows indicate storage into the algorithm's database. The right-hand panel shows the flow for a recovery algorithm in which all evolution is performed on the physical robot.

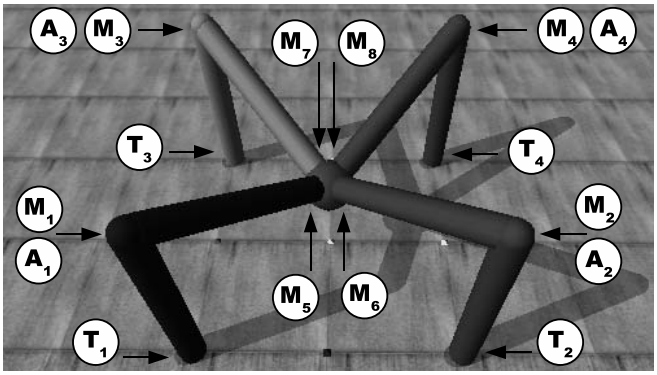

Figure 3. The quadrupedal robot. T_i indicates touch sensors; A_i indicates angle sensors; M_i indicates motorized joints.

With the evolved controller produced Stage I, the 'physical' robot suffers some unanticipated failure or enters a novel environment. The robot is then stopped, and the recent sensor logs are transferred back to the simulator, as well as inserted into the database along with the evolved controller that was controlling the physical robot at that time. During subsequent passes through the algorithm, the robot attempts to function using the compensatory evolved controller produced by Stage I.

Stage II: Damage Hypothesis Evolution. The **diagnostic EA** is used to evolve a hypothesis about the actual failure incurred by the physical robot, or a description of the new environment it has entered. The diagnostic EA uses the sensor logs produced by the physical robot, along with the controllers running on the physical robot at that time, to measure the correctness of each of the diagnoses encoded by the genomes. Once the diagnostic EA has terminated, the most fit damage hypothesis is supplied to the new repair EA, which starts up again in Stage I.

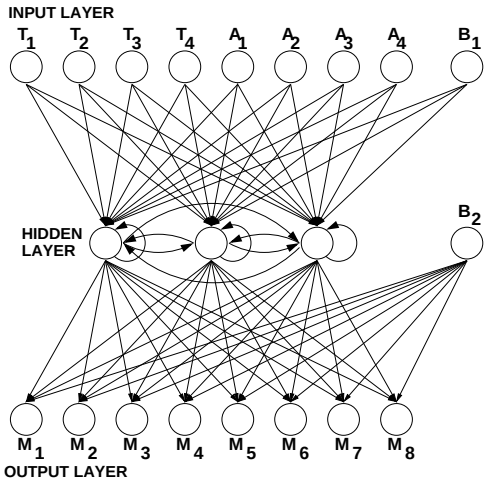

Figure 4. T_i indicates touch sensor neurons; A_i indicates angle sensor neurons; M_i indicates the motor neurons.

2.2 Experimental Setup

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 input layer is fully connected to the hidden layer; each neuron at the hidden layer is fully connected to the output layer, as well as recurrent connections to itself and the other hidden neurons. There are also two bias neurons, one which is fully connected to the hidden layer, and another which is fully connected to the 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.

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 position, orientation and veloc-

ity of the robot based on these torques, along with external forces such as gravity, friction, momentum and collision with the ground plane.

2.3 The Repair EA

The genomes of the repair EA are strings of floating-values, which encode the synaptic weights. There are a total of 68 synapses, giving a genome length of 68 floating-point 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, and each genome specifies the synapses of the simulated robot's neural network controller. If there are any previously evolved controllers stored in the database, these are downloaded into the starting population. The robot is then evaluated in the simulator for 1000 time steps, and the fitness value is returned. 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. Selected genomes undergo mutation: each floating-point value of the copied genome has a 1 per cent chance of undergoing a point mutation. Of the 100 newly generated genomes, 24 pairs are randomly selected and undergo one-point crossover.

The Diagnostic EA. The diagnostic EA evolves hypotheses about the failure incurred by the physical robot, or a description of its new environment. Using only sensory data arriving from the physical robot this EA can distinguish between these two possibilities automatically, as shown in the next section.

For each genome in the diagnostic EA, the simulated robot is broken or the environment is changed according to the genome's diagnosis. The robot is then evaluated using each of the evolved controllers downloaded from the database: during the first pass through the diagnostic EA, there is only a single controller available.

In [5], the function for determining the fitness of genome g_j was set to

$$f(g_j) = \frac{\sum_{i=1}^{c} |f_{\text{act}}(i) - f_{\text{obt}}(i)|}{c}, \qquad (1)$$

where c is the number of controllers evolved so far by the repair EA, $f_{\text{act}}(i)$ is the forward distance travelled by the actual robot using evolved controller i, and $f_{\text{obt}}(i)$ is the forward distance travelled by the simulated robot using evolved controller i after it has been damaged using the damage hypothesis 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 after suffering almost identical failures. This result is not surprising due to the fact that the robot is a highly coupled, highly non-linear system: thus similar initial conditions (two identical robots with identical controllers but only similar failures) are expected to rapidly diverge in behavior over time.

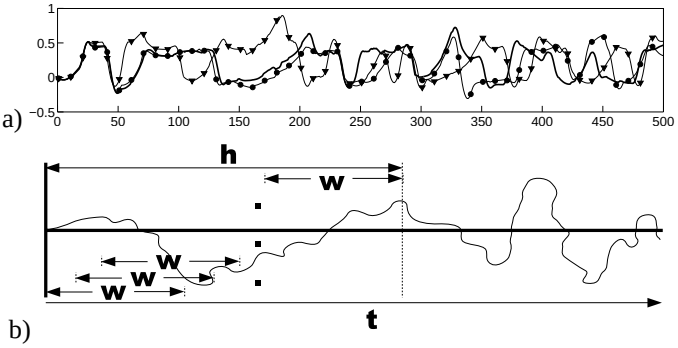

Figure 5. a: The three lines represent the time history of one of the robot's angle sensors using a typical evolved controller when it suffers three different failures. Only the first 500 time steps of the 1000 time step evaluation period are shown. The thick line indicates the sensor values when the 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%. b: Outline of the rolling mean fitness metric.

This point is illustrated by Figure 5a, which shows the time series for one of the robot's proprioceptive sensors when it is controlled by a typical evolved controller. Three time series are shown, correspond to three robot evaluations: in the first one of the robot's motors fails by 10%; in the second trial the same motor fails by 20%; in the third trial one of the touch sensors fails by 50%. As can be seen, the time series of the first two related damages stay correlated for a long period of time (divergence occurs around $t = 150$), while the two unrelated damages diverge early ($t = 50$). Later in the time series 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 repair EA was formulated, in which the fitness of genome p is given by:

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

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

In this formulation i is the number of controller evolved for the actual robot so far; j is the number of sensors contained in the robot; h is some header length, which indicates how

much of initial time series data to use; and w indicates the width of a time window within this time header. $d(i, j, k, p)$ indicates the differences between the senors values of the actual robot and the sensor values of the simulated robot when it is modified by the hypothesis encoded in genome p. $d(i, j, k, p)$ computes a piecewise comparison between short time periods of sensor activation, namely time periods described by $[t - w/2, t + w/2]$. $s_{\text{act}}^{(i,j)}(t)$ indicates the activation of the jth sensor at time t obtained from the actual robot when it is using controller i, and $s_{\text{obj}}^{(i,j,p)}(t)$ indicates the activation of the jth sensor at time t from the hypothesized simulated robot using controller i and described 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 in the database: the controllers evolved by the repair EA in Stage I, and the sensor data attained by the 'physical' robot when using those controllers, respectively (ie. $i > 1$).

The genomes of the diagnostic EA, like the repair EA, are strings of floating-point values. Each genome in the diagnostic EA is composed of a set of four genes: each gene denotes a possible unanticipated situation. There are six different unanticipated situations: a body part breaks off (the joint separates); a body part increases in mass by some amount (from 100% to 200%); a motor weakens to some degree; a sensor fails to some degree; a joint jams to some degree; or the floor on which the robot stands is canted horizontally (tilt ranges between -30 and 30 degrees).

Each of the four genes encoded in the diagnostic EA genomes 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 $[0, 4]$, and indicates which of the five 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.

At the termination of the diagnostic EA the best evolved hypothesis is stored in a database: these hypotheses are used to seed the random population at the beginning of the next run of the diagnostic EA, rather than starting each time with all random hypotheses.

Table 1. Unanticipated Scenarios 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.

For each run of the algorithm described, the repair EA is 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 repair EA, five hardware trials and four passes through the diagnostic EA. Each pass through one of the EAs is computed 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), in which the first passes through the repair and diagnostic EAs began with randomly generated populations.

An additional 130 independent runs were performed as a control: the control algorithm is identical to the algorithm described in the previous section, but rather than using the rolling mean metric described in Equation 3 to evaluate the fitness of a hypothesis, 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 actual algorithm for the same situation.

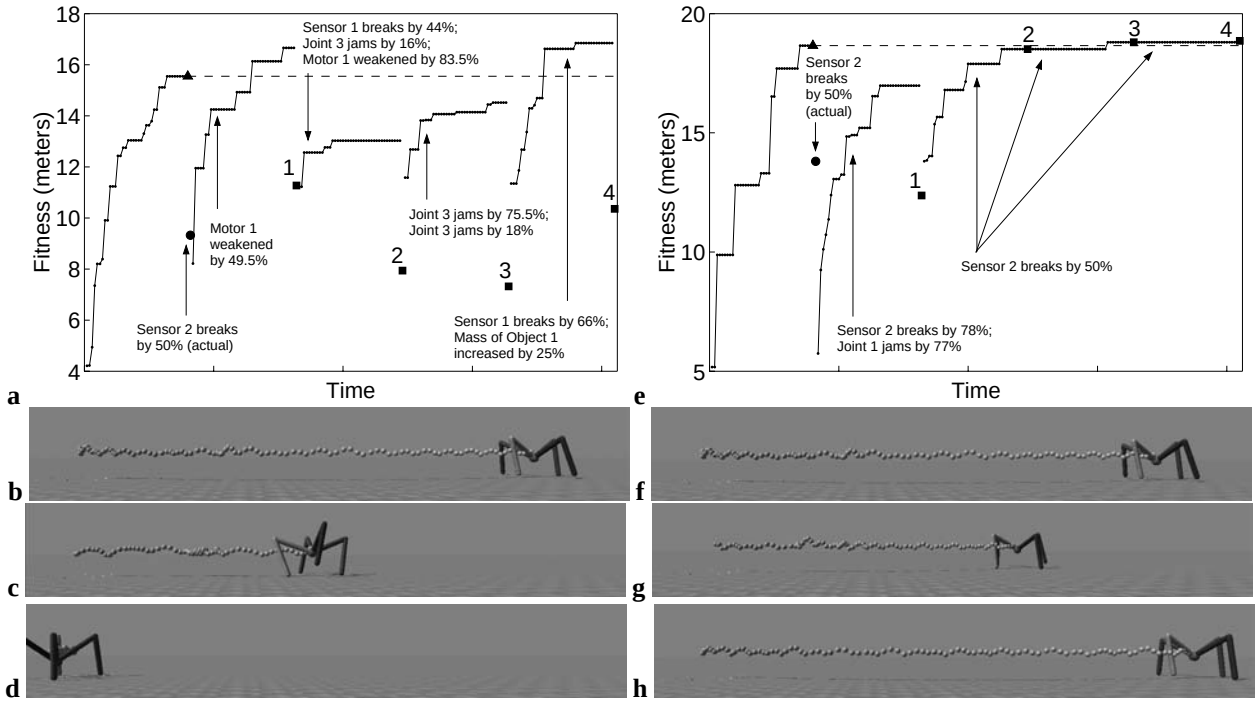

Figure 6. a: The evolutionary progress of the control algorithm for unanticipated situation 5: one of the angle sensors breaks by 50%. The dotted lines indicate the progress of the five passes of the repair EA. The captions indicate the best hypotheses evolved by the four passes through the diagnostic EA. The triangle shows the original fitness (distance travelled in meters) of the physical robot. The circle indicates distance travelled after experiencing the unanticipated situation. The squares indicate the distance travelled by the perturbed 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 actual algorithm for the same situation, but using the rolling mean metric.

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

4 Discussion

Figure 6 shows that the control algorithm, in that instance, was unable to discover the correct hypotheses as to what unanticipated situation the 'physical' robot had encountered. The actual algorithm, on the other hand, guessed a close approximation to the actual situation during the first pass through the diagnostic EA: 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 as well. The second pass through the diagnostic EA, however, aided by the second set of sensor data returned by the 'physical' robot, was able to generate the correct hypothesis. The two final passes through the diagnostic EA retained the correct hypothesis, allowing the repair EA to evolve a controller which actually outperformed the original controller.

Figure 7 makes clear that the actual algorithm far 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 travelled by the 'physical' robot after encountering the situation, and during the fourth hardware trial. However the actual algorithm was able to restore functionality for these five situations successfully. It is notable that these five situation 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 diagnostic EA is small, such as the third unanticipated situation when a leg breaks off, the diagnostic EA can find the correct hypothesis through random search. In order to confirm this explanation, all 130 evolutionary histories generated by the first pass through the diagnostic EA for the control algorithm are plotted in Figure 8a, and the 130 evolutionary histories of the first pass through the diagnostic EA for the actual algorithm are plotted in 8b.

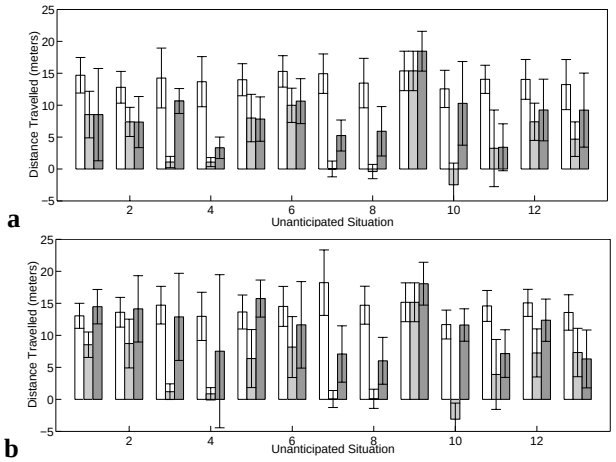

Figure 7. a: The average performance of the control algorithm for all unanticipated situations. b: The average performance of the actual algorithm for all unanticipated situations. The white bars indicate the average distance travelled by the 'physical' robot before encountering the unanticipated situation;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

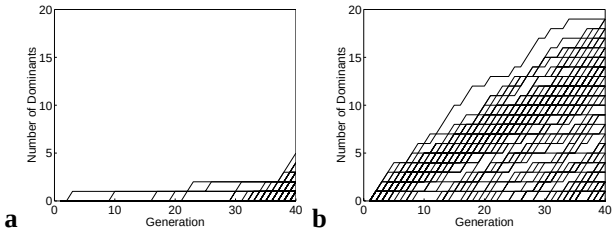

Figure 8. The three lines represent the time history of one of the robot's angle sensors using a typical evolved controller when it suffers three different failures. Only the first 500 time steps of the 1000 time step evaluation period are shown. The thick line indicates the sensor values when the one 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%.

In order to quantify the amount of evolutionary activity in each of these passes, the number of dominants that had appeared in the population so far was recorded during each generation. 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 indicates a population that often moves to better regions of the search space; one that produces few dominants does not make any evolutionary progress, but continues to search those regions of the search space pinpointed by the original, randomly generated genomes.

Figure 8 makes clear that the diagnostic EA of the actual algorithm hosts many dominants, indicating that there is much evolutionary activity. In contrast to this the diagnostic EA displays little evolutionary activity. Because the only difference between the diagnostic EAs in the two algorithms is the fitness function, we can safely conclude that the new fitness metric introduces many gradients into the fitness landscape, allowing the actual 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 repair EA: 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). This was probably due to the drastic effect on all sensors induced by this change. It is important to notice that our algorithm can distinguish between internal damage and external environmental change based solely on changes in distance travelled (in the control algorithm) or sensory data (in the actual algorithm).

Finally, the last three unanticipated situations are truly unanticipated: the diagnostic EA can only approximate the situation using combinations of the five encoded situation types. Not surprisingly, both the control and actual algorithm have difficulty evolving a compensatory controller for these situations. However, the actual 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 described by combinations of encoded situations.

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 sit-

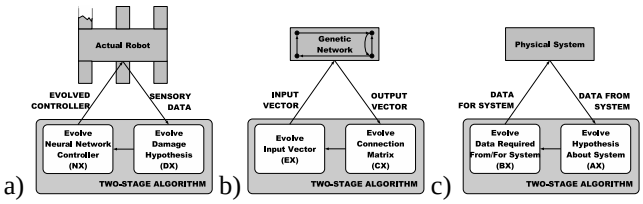

Figure 9. Instantiations of the general two-stage algorithm. a: The robot application described in this paper. b: An application to gene network inference. c: General application to a non-linear system.

uation, without requiring extensive hardware testing. In this paper we have described a two-stage evolutionary robotics algorithm that can automatically diagnosis and recover from a wide range of unanticipated internal damage or external 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.

Here we have introduced a new fitness metric which better allows our algorithm to evolve a correct description of the unanticipated situation encountered by the robot. This metric has been shown to allow the algorithm to autonomously distinguish between internal damage and external environmental 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. Elsewhere we have applied our algorithm to the problem of gene network inference; a schematic of the generalized algorithm is shown in Figure 9. Future avenues of study will include applying our robot-based algorithm to an actual physical robot, as well as generalizing the algorithm to a wide range of coupled, non-linear systems.

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. Automated damage diagnosis and recovery for remote robotics.

[6] 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.

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

[8] 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.

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

[10] 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.

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

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

[13] 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, Spinger, Berlin, 2003.

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

[15] `opende.sourceforge.net`

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

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

[18] 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.