Automated Damage Diagnosis and Recovery for Remote Robotics

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

***Abstract*— Remote robotics applications, such as space exploration or operation in hazardous environments, would greatly benefit from automated recovery algorithms for unanticipated failure or damage. In this paper a two-stage evolutionary algorithm is introduced that forwards this aim by first evolving a damage hypothesis after failure and then re-evolving a compensatory neural controller to restore functionality. The algorithm pre-supposes that a continuous robot simulator is running onboard the physical robot; In this paper, the 'physical' robot is also simulated, but in future work the algorithm will be applied to a real, physical robot. Although evolutionary algorithms require a large number of evaluations to produce a useful solution, our preliminary results indicate that almost complete functionality can be restored after only three evaluations on the 'physical' robot, as opposed to over 3000 evaluations if the compensatory controller is evolved all on the physical robot. Our algorithm also has the benefit of producing a diagnostic model of the failure.**

I. Introduction

Recovery from error, failure or damage is a major concern in robotics. A majority of effort in programming automated systems is dedicated to error recovery [18]. The need for automated error recovery is even more acute in the field of remote robotics, 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.

A number of algorithms based on repeated testing have been proposed and demonstrated for robotics [8], [3], and electronic circuits [5], [16]. 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.

In this paper, an offline damage diagnosis and repair algorithm is proposed, which, for the results presented here, requires only three hardware trials to restore (on average) complete functionality.

Srinivas [15] was one of the first researchers to study error diagnosis and recovery, but his approach, along with subsequent approaches ([6], [1], [7], [9], [17]), 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 [10] for error recovery. However their algorithm also only handles prespecified error types. The algorithm proposed here is demonstrated to automatically provide an approximate diagnosis and successfully recover from an unanticipated failure type.

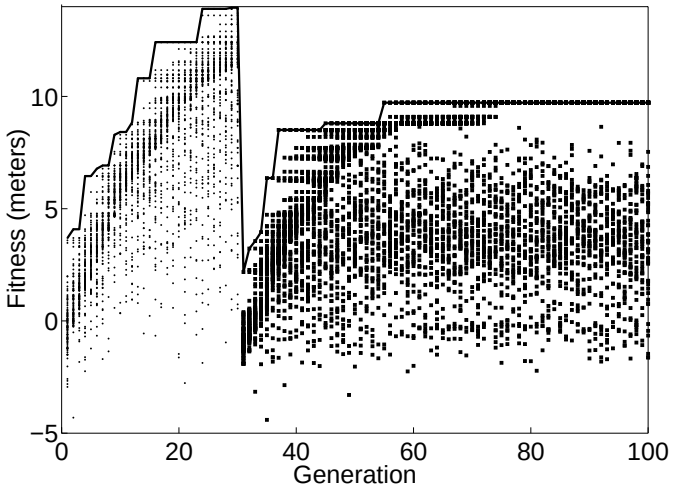

Fig. 1. **Results of a standard evolutionary recovery.** The evolutionary history for an evolving robot population. Controllers for forward locomotion are evolved on the simulated quadrupedal robot (Figure 3a) for the first 30 generations (each dot represents one or more simulated robot evaluations). 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.

Mahdavi and Bentley [8] 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 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 [14], [11], [2], [4]. Here we also use evolutionary algorithms to co-evolve bodies and brains, but use an inverse process: instead of evolving a controller given a morphology, we evolve a morphology given a controller, and instead of evolving to reach a high fitness as a form of design, we evolve towards an observed low fitness as a form

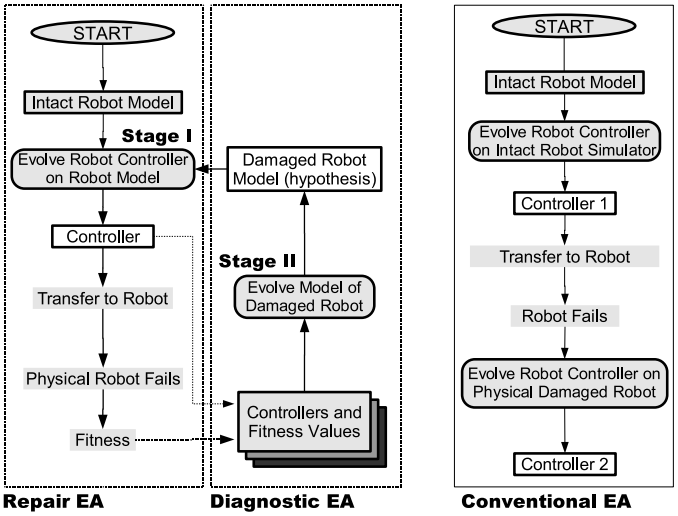

Fig. 2. **Two algorithms for evolved function recovery.** The two left-hand panels show the flow for the recovery algorithm proposed in this paper. The dotted arrows indicate storage into the algorithm's database. The right-hand panel shows the flow for a recovery algorithm in which all evolution is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

of diagnosis. By avoiding a distinction between the robot's morphology or controller, and by employing an evolutionary algorithm, the algorithm can compensate for damage or failure of the robot's mechanics, its sensory or motor apparatus, or the controller itself, or some combination of these failure types. 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.

Moreover, by using an evolutionary algorithm for recovery, qualitatively different behaviours (such as hopping instead of walking) evolve in response to drastic failure (such as the loss of an entire leg). More traditional analytic approaches can only produce parametrically modified behaviours in response to mild damage.

II. Methodology

Algorithm Overview

The algorithm for failure recovery has two stages: controller evolution and damage hypothesis evolution. The algorithm also maintains a database, which stores pairs of data: an evolved controller and the fitness produced by the physical robot when that controller is used. Two separate evolutionary algorithms—the repair EA and the diagnostic EA–are used to generate controllers for the simulated and physical robot, as well as hypotheses regarding the failure incurred by the physical robot, respectively. 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 damaged physical robot, using the current best damage hypothesis generated by Stage II. When the repair EA terminates, the best controller from the run is transferred to and used by the physical robot.

Physical Robot Failure. The physical robot uses an evolved controller to walk forwards. An unanticipated failure occurs to the robot, and the broken robot records its own forward displacement for a period of time. The physical robot is then stopped, and the recorded fitness is transferred back to the simulator, as well as inserted into the database along with the evolved controller on-board the robot at that time. During subsequent passes through the algorithm, the damaged 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. The diagnostic EA uses the fitness values produced by the broken 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.

Experimental Setup

We applied the proposed algorithm to the recovery of locomotion of severely damaged legged robots. In these experiments, the 'physical' robot was also simulated.This is achieved by using two separate evolutionary algorithms (EAs) to evolve either a controller or damage hypothesis in simulation: these two EAs are referred to as the repair and diagnostic EAs, respectively. The simulation runs on board the 'physical' robot, which in this paper is also simulated: evolved controllers are uploaded from the simulation to the physical robot, and performance measurements are downloaded from the physical robot to the simulation. The robot simulator is based on Open Dynamics Engine, an open-source 3D dynamics simulation package [12]. The simulated robot is composed of a series of three-dimensional objects, connected with one-degree of freedom rotational joints.

The Robots. The two hypothetical robots tested in this preliminary work—a quadrupedal and hexapedal robot—are shown in Figure 3.

The quadrupedal robot has eight mechanical degrees of freedom: two one-degree-of-freedom rotational joints per leg, one at the shoulder, and one at the knee. The quadrupedal robot contains four binary touch sensors, one in each of the lower legs: the sensor fires if the lower leg is one the ground, and doesn't fire otherwise. There are also four angle sensors in the shoulder joints, which return a signal commensurate with the flex or extension of that joint. Each of the eight joints is actuated by a torsional motor. The joints have a maximum extension of 30 degrees from their original setting (shown in Figure 3), and a maximum flex of 30 degrees. The hexapedal robot has 18 mechanical degrees of freedom: each leg has a one-degree-of-freedom rotational joint at the knee, and one two-degree-of-freedom rotational joints connecting the leg to the spine. Each joint is actuated by a torsional motor, and the joint ranges are the same as for the quadrupedal robot. The hexapedal robot contains six touch sensors, one per lower leg, and six angle sensors, placed on the joints connecting the legs to the spine.

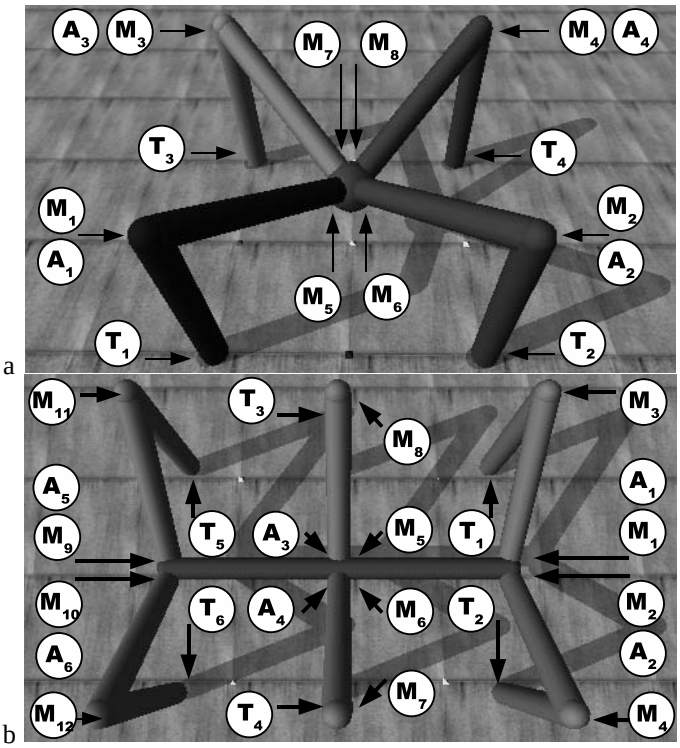

Fig. 3. **The simulated robots used for experimentation. a:** The quadrupedal robot. **b:** The hexapedal robot. T_i indicates touch sensors; A_i indicates angle sensors; M_i indicates motorized joints.

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 rotation, $\frac{\pi}{2}$. 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 velocity of the robot based on these torques, along with external forces such as gravity, friction, momentum and collision with the ground plane.

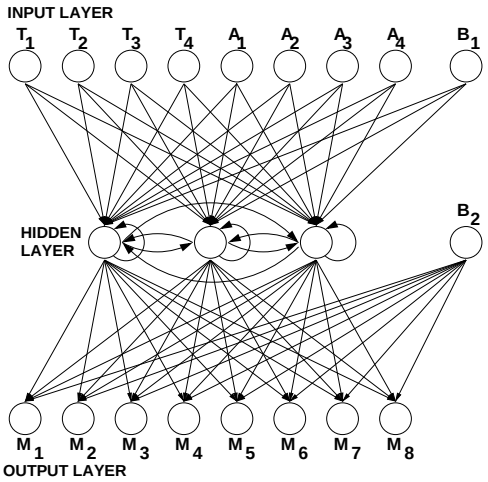

Fig. 4. **The neural network architecture used for the quadrupedal robot.** T_i indicates touch sensor neurons; A_i indicates angle sensor neurons; M_i indicates the motor neurons. The hexapedal robot's network architecture is slightly larger: it contains 10 sensor and 12 motor neurons. (Figure 3).

Algorithm Implementation

The Repair EA. The repair EA is used to generate sets of synaptic weights for the robot's neural network (Figure 4).

The fitness function rewards robots for walking forwards as far as possible during 1000 time steps of the simulation. The fitness function is given as $f(g_i) = d(t_{1000})$ - $d(t_1)$, where $f(g_i)$ is the fitness, measured in meters, of the robot containing a neural network with synapses labelled using the values encoded genome g_i. $d(t_1)$ is the forward displacement of the robot, measured in meters, at the first time step of the simulation; $d(t_{1000})$ is the forward displacement of the robot (again in meters) at the final time step of the simulation.

The genomes of the repair EA are strings of floating-values, which encode the synaptic weights. For the quadrupedal robot, there are a total of 68 synapses, giving a genome length of 68 floating-point values. For the hexapedal robot, there are a total of 120 synapses, giving a genome length of 120 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 100 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 50 least fit genomes are deleted from the population. Fifty new genomes are selected to replace them from the remaining 50, 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 50 newly generated genomes, 12 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.

For each genome in the diagnostic EA, the simulated robot is initially broken according to the genome's diagnosis, and

the broken robot is then evaluated using each of the evolved controllers downloaded from the database. The fitness function is an attempt to minimize the difference between the simulated and physical robots' fitness values. This is based on the observation that the closer the damage hypothesis encoded in the genome is to the actual damage, the lesser the difference between the two behaviours.

During subsequent passes through this stage, there are additional pairs of evolved controllers and fitness values in the database: the controllers evolved by the repair EA in Stage I, and the fitness values attained by the 'physical' robot when using those controllers, respectively. In these cases, the simulated robot is evaluated once for each of the evolved controllers, and the fitness of the genome is then the sum of the errors between the predicted and observed fitness values.

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 damage. In this preliminary study, the actual robot can undergo three different types of damage—joint breakage, joint jamming, and sensor failure—and can incur zero to four of these damages simultaneously. In joint breakage, any single joint of the robot can break completely, separating the two parts of the robot connected by that joint. In joint jamming, the two objects attached by that joint are welded together: actuation has no effect on the joint's angle. In sensor failure, any sensor within the robot (either one of the touch or angle sensors) feeds a zero signal into the neural network during subsequent time steps. Any type of failure that does not conform to one of these types is referred to henceforth as an unanticipated failure: in order to compensate for such cases, the diagnostic EA has to approximate the failure using aggregates of the encoded failure types.

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 values. 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. If the gene is dormant, the damage encoded by this particular gene is not applied to the simulated robot during evaluation. If the gene is active, the second floating-point value is rounded to an integer in $[0, 2]$, and indicates which of the three damage types should be applied to the simulated robot. If the damage type is either joint breaking or joint jamming, the third value is scaled to an integer in $[0, j-1]$, where j is the number of mechanical degrees-of-freedom of the robot ($j = 8$ for the quadrupedal robot, and $j = 12$ for the hexapedal robot). If the damage type is sensor failure, then the third value is scaled to an integer in $[0, s-1]$, where s is the total number of sensors contained in the robot ($s = 8$ for the quadrupedal robot and $s = 12$ for the hexapedal robot). The fourth value is currently not used in this preliminary study, but will be used for additional damage types that are not binary, but occur with a lesser or greater magnitude (ie., a sensor that experiences 80% damage, instead of completely failing).

The diagnostic EA is similar to the repair EA but the fitness criterion here is the ability to match the observed low fitness.

TABLE I
DAMAGE CASES TESTED

Case	Explanation
1	One of the lower legs breaks off.
2	One of the entire legs breaks off.
3	One of the touch sensors fails.
4	One of the entire legs breaks off, and the touch sensor on one of the intact legs fails.
5	One of the angle sensors fails.
6	One of the upper-leg joints jams.
7	One of the entire legs breaks off, and one of the upper-leg joints jams on an intact leg.
8	One of the entire legs breaks off, one of the upper-leg joints jams on an intact leg, and one of the angle sensors fails on another intact leg.
9	Nothing breaks.
10	One of the hidden neurons fails.

Also, 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.

III. RESULTS

A control case experiment was first performed which conforms to the algorithm outlined in the right-hand panel of Figure 2: all evolution is performed on the 'physical' robot after damage. The results of this run are shown in Figure 1. In this case, controller evolution is performed by an instantiation of the repair EA until generation 30 on the quadrupedal robot. The controller is then transferred to the 'physical' robot, which then undergoes separation of one of its lower legs (damage case 1). The repair EA is then re-started, but works on the 'physical' robot directly.

The algorithm proposed in this paper was then applied to both the quadrupedal and hexapedal robots, each of which suffered 10 different damage possibilities: the 10 cases are listed in Table I. For each run of the algorithm, the repair EA is run once to generate the initial evolved controller, and then both the repair and diagnostic EAs are run three times each after physical robot failure. Each EA is run for 30 generations, using a population size of 100 genomes. A total of 20 runs of the algorithm were performed (10 damage cases for each of the two robots), in which both the repair and diagnostic EAs were initialized with independent random starting populations.

Cases 1, 2, 3, 5 and 6 are simple damage cases: they can be described by a single gene in the genomes of the diagnostic EA. Cases 4, 7 and 8 represent compound damage cases, and require more than one gene to represent the damage. Case 9 represents the case when the physical robot signals that it has incurred some damage, when in fact no damage has occurred. Case 10 represents an unanticipated failure: hidden neuron failure cannot be described by the diagnostic EA genomes.

For each case, the repair and diagnostic EAs were started with independent initial random populations. Each case was run until three evaluations had been performed on the 'physical' robot, after the initial failure. Figure 5 shows the recovery of the quadrupedal robot after minor damage (case 3) and after unanticipated damage (case 10), and recovery of the hexapedal robot after severe, compound damage (case 8). The recovery of both robots after undergoing all 10 damage cases is shown in Figure 6.

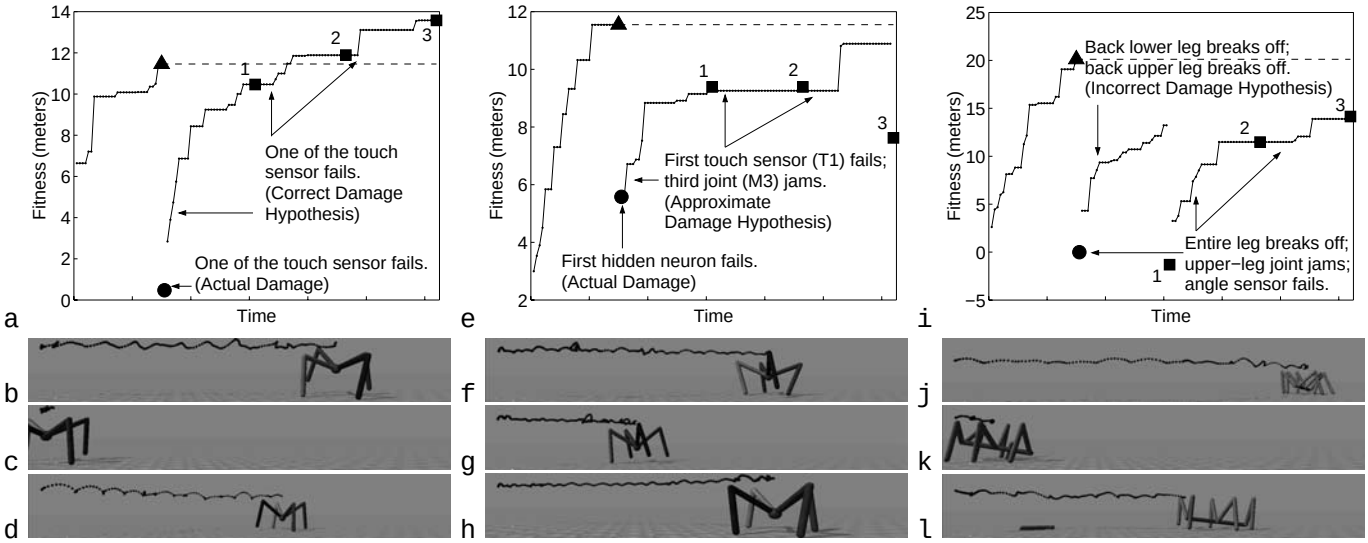

Fig. 5. **Three typical damage recoveries. a**: The evolutionary progress of the four sequential runs of the repair EA on the quadrupedal robot, when it undergoes a failure of one of its touch sensors. The hypotheses generated by the three runs of the diagnostic EA (all of which are correct) are included. The small circles indicate the fitness of the best controller at each generation of the repair EA. The triangle shows the fitness of the first evolved controller on the physical robot (the behaviour of the 'physical' robot with this controller is shown in **b**); the large circle shows the fitness of the robot after the damage occurs (the behaviour is shown in **c**); the squares indicate the fitness of the physical robot for each of the three hardware trials (the behaviour of the physical robot during the third trial is shown in **d**). **e-h** The recovery of the quadrupedal robot when it experiences unanticipated damage. **i-l** The recovery of the hexapedal robot when it experiences severe, compound damage. The trajectories in **b-d**, **f-h** and **j-l** show the change in the robot's centre of mass over time (the trajectories are displaced slightly upwards for the sake of clarity).

IV. Analysis

As can be seen in Figure 1, even after 70 generations have elapsed after the physical robot suffers damage for the control case, and 3550 hardware evaluations have been performed, total function has not been restored. The degree of restoration (about 70%) is less than the degree of restoration when the proposed algorithm is applied to the quadrupedal robot undergoing the same type of damage (the leftmost bars in Figure 6a). However the proposed algorithm only requires three hardware evaluations, or two orders of magnitude fewer hardware trials.

Figure 5 shows that for these three cases, much functionality is restored to the 'physical' robot after only three hardware trials. Indeed, in the case of sensor failure for the quadrupedal robot, the fitness of the physical robot after the third hardware trial exceeds its original functionality. Often, the compensatory controller produces a much different gait from that exhibited by the original, undamaged robot. For example the robot enduring sensor failure exhibits a hopping gait, as evidenced by the discrete arcs in the trajectory of its centre of mass (Figure 5d), compared to a more stable but erratic gait for the undamaged robot (Figure 5b).

Figure 6a indicates that functionality is restored for all but the severe compound damage cases (cases 7 and 8), and for all but three of the damage cases for the hexapod (cases 4, 7 and 10). In these cases, the algorithm never converged on the correct damage hypothesis. (Of course, this is to expected for case 10, in which it is impossible for the diagnostic EA to converge on the correct hypothesis.)

It is believed that the reason for the sporadic failure of the algorithm is due to the information-poor method of comparing the simulated robot's behaviour against the physical robot's behaviour, which in this paper is done by simply comparing forward displacement. This method will be replaced in future with a more sophisticated method such as comparing sensor time series or measuring the differences in gait patterns.

Figure 6 also indicates that the algorithm performs equally well for both morphological and controller damage: function recovery for both cases 1 and 2 (morphological damage) and cases 3 and 5 (controller damage), for both robots, approaches or exceeds original performance. Indeed, it would be straightforward to generalize this algorithm beyond internal damage to incorporate adaptive response to environmental change:the diagnostic EA could evolve not only internal damage hypotheses but also hypotheses regarding environmental change, such as increased ruggedness of terrain or high winds.

Unanticipated Failures. Most importantly, the quadrupedal robot is able to recover functionality when an unanticipated failure occurs: the failure of a hidden neuron (case 10). This type of failure cannot be encoded in the diagnostic EA, but the EA is able to approximately describe the failure with aggregates of other damage types, in this case the failure of a touch sensor (T_1) and the jamming of one of its lower leg joints (M_3) (Figure 5e). Table II indicates a subset of the synaptic weights for the original evolved controller for this particular case (this controller produces a forward displacement of approximately 11.5 meters in the undamaged 'physical' robot). As can be seen, the first hidden neuron has a very strong influence on the third joint, and less influence on the other joints.

Also, the third joint is influenced more by the first hidden neuron than by the other two hidden neurons. Thus, when the first hidden neuron fails, the third motor neuron will output values near zero, and the motor will work to keep its joint at the starting, default value. This explains why approximating the hidden neuron failure with a jammed third joint produces a close approximation of the actual damage. Future experiments

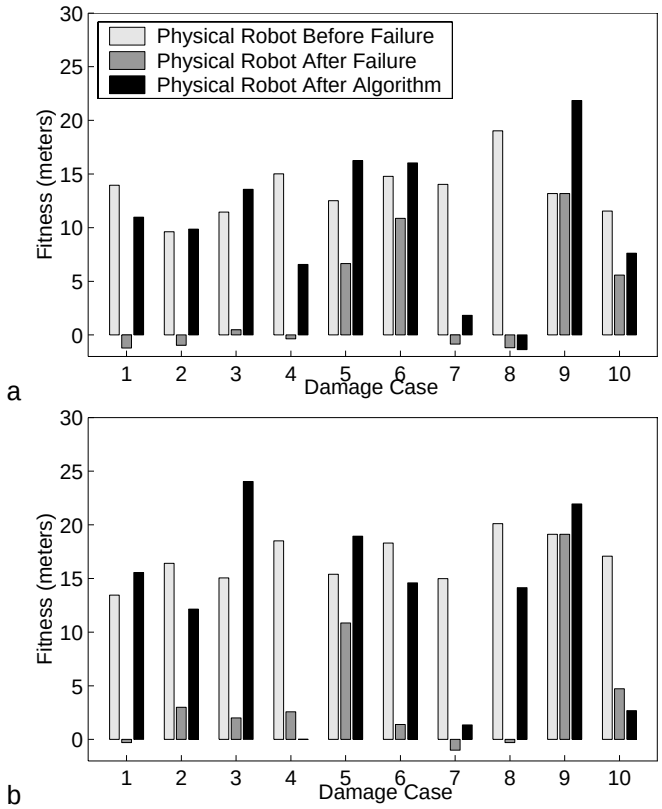

Fig. 6. **Results for all of the damage cases. a:** The recovery of the quadrupedal robot for all 10 damage cases. The light gray bars indicate the fitness of the physical robot before failure; the medium gray bars indicate fitness after failure; and the black bars indicate fitness after the third hardware trial. **b:** The recovery of the hexapedal robot for all 10 damage cases.

are planned to investigate how unanticipated failures can be spanned by aggregates of anticipated damage types.

V. Conclusions

In this paper an evolutionary approach to damage diagnosis and recovery has been proposed. This algorithm is the first of its kind in which there is continuous, automated bidirectional information flow between the simulation and the physical robot: the physical robot provides information about its current state, which is used to update the state of the simulator, and the simulator provides neural controllers for the undamaged or damaged physical robot.

This algorithm has several advantages. First, the algorithm requires a minimum of hardware trials on a physical robot, effectively reducing the number of hardware trials by two orders of magnitude, compared to an algorithm that evolves recovery all on the physical robot. Second, the algorithm does not distinguish between morphological damage or controller failure, and could easily be generalized to recover from internal damage and respond to external environmental changes. Third, in some cases the algorithm can provide recovery from compound damage. Fourth, if the simulation which produces hypotheses about the state of the physical robot and compensatory controllers were run onboard the physical robot, function recovery and adaptation to the environment could be continuous. Finally, in some cases the algorithm can provide recovery for unanticipated damage or failure.

TABLE II
Synaptic Weight Matrix for an Evolved Controller

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
H_1	-0.33	-0.68	**-0.99**	-0.39	-0.55	-0.54	+0.31	-0.95
H_2			-0.51					
H_3			+0.33					

Future experiments are planned in which the information sent back into the simulator by the physical robot—candidates include movement data and sensor time series—is improved, such that the diagnostic EA can more easily generate damage hypotheses. Also, environmental factorswill be incorporated into the diagnostic EA so that the robot can not only recover from internal damage, but respond to environmental change.

It is currently assumed that evolved controllers transferred from simulation to the 'physical' robot produce identical behaviour: in future the transferral will be made more realistic by first adding noise to the behaviour of the 'physical' robot, and finally by replacing the (currently simulated) physical robot with a real physical robot.

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] 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.

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

[7] 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.

[8] 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.

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

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

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

[12] `opende.sourceforge.net`

[13] N. Roy and S. Thrun. Online self-calibration for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 3:2292–2297, 1999.

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

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

[16] 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.

[17] 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.

[18] M.C. Zhou and F. DiCesare. Adaptive design of Petri-Net controllers for error recovery in automated manufacturing systems. In *IEEE Trans. on Systems, Man and Cybernetics*, 19(5), pp. 963–973, 1989.