

Automated Damage Diagnosis and Recovery for Remote Robotics

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

Hod Lipson

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—which we call the *estimation-exploration algorithm*—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 presupposes that a robot simulator is running continuously 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, the results reported here 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 [19]. 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. The recent difficulties with NASA’s Spirit and Opportunity Mars rovers provide a dramatic example; both robots suffered different, unanticipated partial failures [13]. 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 for error recovery have been proposed and demonstrated for both robotics [8], [3], and electronic circuits [5], [10]. However, repeated generate-and-test algorithms for robotics are 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. 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 [17] was one of the first researchers to study error diagnosis and recovery, but his approach, along with subsequent approaches ([6], [1], [7], [9], [18]), required online operation (repeated testing on the physical robot), and could not handle unanticipated errors.

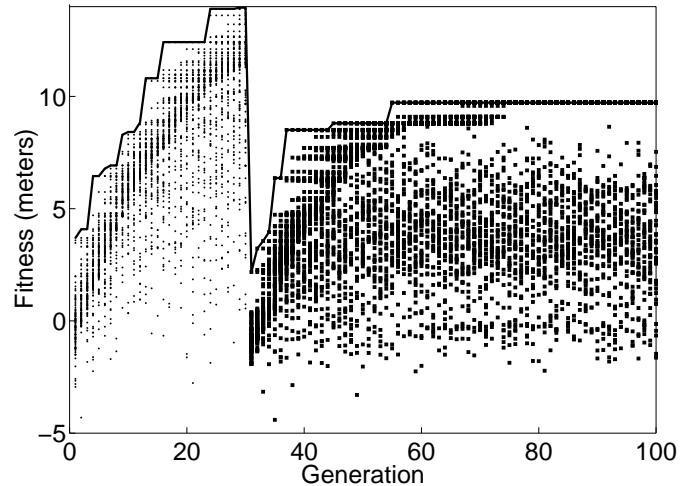


Fig. 1. **Results of a standard evolutionary recovery.** The evolutionary history for an evolving robot population. Controllers for forward locomotion are evolved on the simulated quadrupedal robot (Figure 3a) for the first 30 generations (each dot represents one or more simulated robot evaluations).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

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 only handles those failures which are most likely to occur, given the particular robot system: likely failures are first proposed, and recovery logic for those failures is generated. By definition, this approach can only handle anticipated failures. However the algorithm proposed here is shown to automatically provide an approximate diagnosis of some unanticipated failure, as well as recover from it, while the robot is performing its task. This failure may not necessarily be likely, or even have been previously experienced by the robot.

Mahdavi and Bentley [8] recently demonstrated the ability of an online evolutionary algorithm to automatically recover 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 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 the morphology and the controller of simulated robots together in order to achieve some desired behavior [16], [12], [2], [4]. Here we also use evolutionary algorithms to co-evolve robot bodies and brains, but use an inverse process: instead of evolving a controller given a robot morphology, we evolve a robot morphology given a controller. Also, instead of evolving to reach a high fitness as a form of design, we evolve towards an observed low fitness (caused by some unknown failure) as a form of diagnosis. By not making 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 a few pre-specified failures.

Moreover, by using an evolutionary algorithm for recovery, qualitatively different behaviors (such as hopping instead of walking) evolve in response to failure. More traditional analytic approaches can only produce slightly modified behaviors in response to mild damage.

II. METHODOLOGY

Algorithm Overview

The estimation-exploration algorithm has two functions: damage hypothesis evolution (the estimation phase) and controller evolution (the exploration phase). 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 estimation EA and the exploration EA—are used to generate hypotheses regarding the failure incurred by the physical robot, as well as controllers for the simulated and ‘physical’ robot, respectively. Figure 2 outlines the flow of the algorithm, along with a comparison against an algorithm for evolved function recovery all on a physical robot.

Exploration Phase: Controller Evolution. The exploration 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 phase 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 the estimation phase. When the exploration 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 forward displacement (fitness) is inserted into the database along with the evolved controller on-board the robot at that time: these become an input-output pair used to reverse engineer the damage suffered by the robot. During subsequent passes through the algorithm, the damaged robot attempts to function using the compensatory evolved controller produced by the exploration phase.

Estimation Phase: Damage Hypothesis Evolution. The estimation EA is used to evolve a hypothesis about the actual failure incurred by the physical robot. The estimation EA uses the forward displacements produced by the broken physical robot, along with the corresponding controllers running on

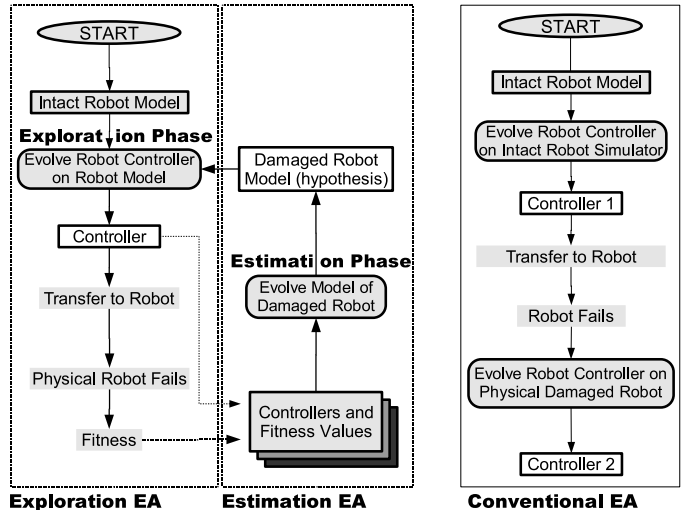


Fig. 2. Two algorithms for evolved function recovery. The two left-hand panels show the flow for the exploration-estimation algorithm, applied to robot recovery. The dotted arrows indicate storage into the algorithm’s database. The right-hand panel shows the flow for a recovery algorithm in which all evolution is performed on the physical robot.

the physical robot at that time, to measure the correctness of each of the diagnoses encoded by the estimation EA’s genomes. When the estimation EA terminates, the most fit damage hypothesis is supplied to the exploration EA. The robot simulator is updated to model this damage hypothesis: for example if the hypothesis is that one of the legs has fallen off, that leg is broken off of the simulated robot. The exploration EA then evolves a compensatory controller using this updated model.

Experimental Setup

We applied the proposed algorithm to the recovery of locomotion of severely damaged legged robots. A robot simulator is used to evolve controllers for the ‘physical’ robot: in this paper the ‘physical’ robot 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 [14]. 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. There are 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 touch sensor returns 1.0 if the lower leg is on the ground, and -1.0 otherwise. There are also four angle sensors in the shoulder joints, which return a signal commensurate with the flex or extension of that joint (-1.0 for maximum flexure up to 1.0 for maximum extension). Each of the eight joints is actuated by a torsional motor. The joints have a maximum flex of 30 degrees from their original setting (shown in Figure 3), and a maximum extension of 30 degrees. The hexapedal robot has 18 mechanical degrees of freedom: each leg has a one degree-of-freedom rotational joint

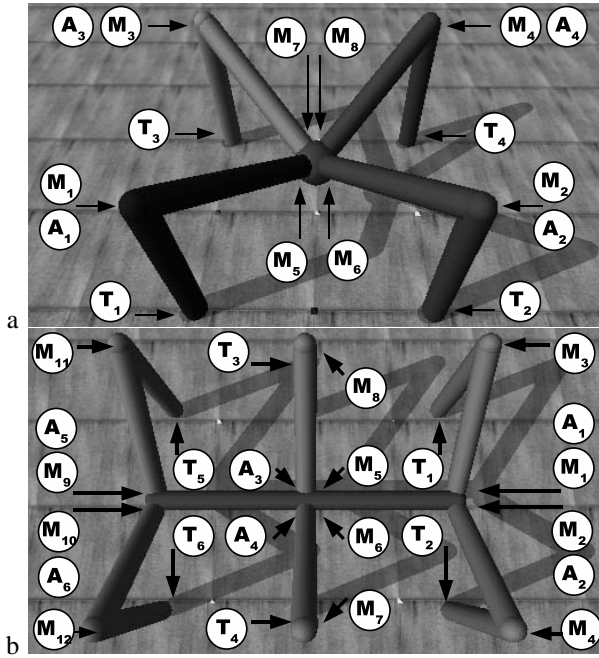


Fig. 3. **The simulated robots used for experimentation.** a: The quadrupedal robot. b: The hexapedal robot. T_i indicates touch sensors; A_i indicates angle sensors; M_i indicates motorized joints.

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.

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, and finally propagates the signals to an output layer. The neural network architecture and connectivity is shown in Figure 4. Neuron values and synaptic weights are scaled to 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 torques. 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.

Algorithm Implementation

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

The fitness function rewards robots for moving 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 $f(g)$ is the fitness, measured in meters, of the robot whose neural network controller is labelled with the values encoded

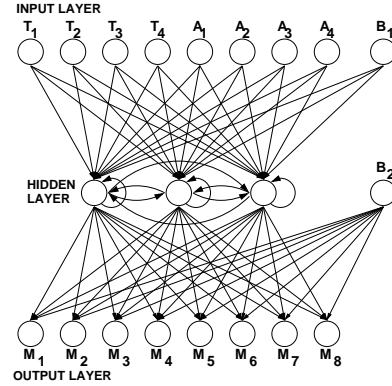


Fig. 4. **The neural network architecture used for the quadrupedal robot.** T_i indicates touch sensor neurons; A_i indicates angle sensor neurons; M_i indicates the motor neurons. The hexapedal robot's network architecture is slightly larger: it contains 10 sensor and 12 motor neurons. The controllers for both robots however contain three hidden neurons, and two bias neurons (B_1 and B_2) which always output a value of 1. (The locations of the sensors and motors on the robots are shown in Figure 3).

in genome g . $d(t_1)$ 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 exploration EA are strings of floating-point values, which encode the synaptic weights. For the quadrupedal robot, there are a total of 68 synapses, giving a genome length of 68. For the hexapedal robot, there are a total of 120 synapses, giving a genome length of 120. 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. If there are any previously evolved controllers stored in the database, these are downloaded into the starting population. A genome is evaluated as follows: the encoded weights are used to label the controller; the robot is then evaluated in the simulator for 1000 time steps using that controller; and the resulting 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 Estimation EA. The estimation EA evolves hypotheses about the failure incurred by the physical robot.

The genomes of the estimation EA, like the exploration EA, are strings of floating-point values. Each genome in the estimation EA is composed of four genes: each gene denotes a possible failure. 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 estimation EA has to approximate the failure using aggregates of the encoded failure types.

Each of the four genes encoded in the estimation EA genomes is comprised of four floating-point values, giving a total genome length of 16 values. Like the exploration 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 breakage 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).

For each genome in the estimation EA, the simulated robot is initially broken according to the failure scenario encoded in the genome, and the broken robot is then evaluated using the controller just evolved by the exploration EA and tested on the ‘physical’ robot.

The fitness function for the estimation EA is an attempt to minimize the difference between the forward displacement achieved by the ‘physical’ robot using that controller, and the forward displacement achieved by the simulated robot using the encoded damage hypothesis. 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 behaviors.

During subsequent passes through the estimation phase, there are additional pairs of evolved controllers and forward displacements in the database: the controllers evolved by the exploration EA 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 forward displacements.

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

The estimation EA is similar to the exploration EA, except for the length of the genomes, what those genomes encode, and the fitness function: in the exploration EA, forward displacement is maximized; in the estimation EA, error between the simulated and ‘physical’ robots’ forward displacements is minimized.

TABLE I
DAMAGE SCENARIOS TESTED

Scenario	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.

III. RESULTS

A control 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 the exploration 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 exploration EA then continues on the ‘physical’ robot for a further 70 generations.

The algorithm proposed in this paper was then applied several times to the quadrupedal and hexapedal robots. During each application of the algorithm, the robots suffered a different damage scenario: the 10 scenarios are listed in Table I.

For each run of the algorithm, the exploration EA is run once to generate the initial evolved controller, and then both the estimation and exploration EAs are run three times each after physical robot failure. Each EA is run for 30 generations, using a population size of 100 genomes. Twenty runs of the algorithm were performed (10 damage cases for each of the two robots), in which both the exploration and estimation EAs were initialized with independent random starting populations seed with any previously evolved controllers or damage hypotheses.

Damage scenarios 1, 2, 3, 5 and 6 can be described by a single gene in the genomes of the estimation EA. Scenarios 4, 7 and 8 represent compound failures, and require more than one gene to represent them. Case 9 represents the situation 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 estimation EA genomes.

Figure 5 shows the recovery of the quadrupedal robot after minor damage (scenario 3); after suffering unanticipated damage (scenario 10); and recovery of the hexapedal robot after severe, compound damage (scenario 8). The recovery of both robots for all 10 damage scenarios is shown in Figure 6.

IV. ANALYSIS

As can be seen in Figure 1, even after 70 generations have elapsed after the ‘physical’ robot suffers damage for the control experiment, and 3550 hardware evaluations have been performed, total function has not been restored. The degree of restoration (about 70%) is about the same as that achieved by the quadrupedal robot suffering the same type of damage

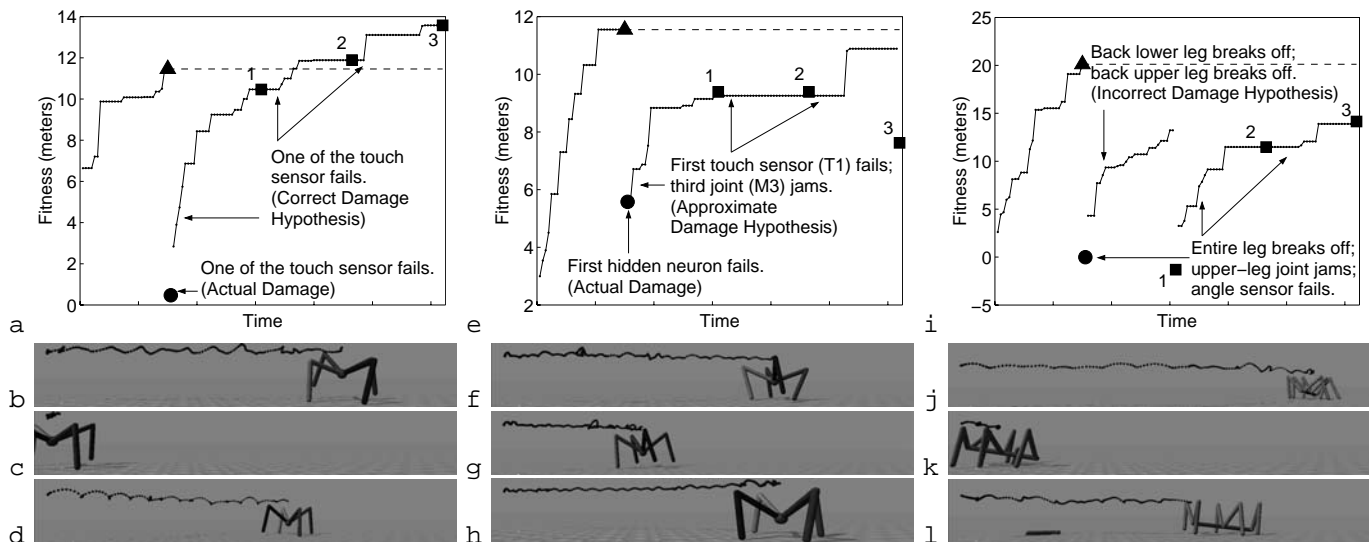


Fig. 5. **Three typical damage recoveries.** **a**: The evolutionary progress of the four sequential runs of the exploration EA on the quadrupedal robot, when it undergoes a failure of one of its touch sensors. The hypotheses generated by the three runs of the estimation EA (all of which are correct) are included. The dots indicate the fitness of the best controller from each generation of the exploration EA. The triangle shows the fitness of the first evolved controller on the ‘physical’ robot (the behavior of the ‘physical’ robot with this controller is shown in **b**); the filled circle shows the fitness of the robot after the damage occurs (the behavior is shown in **c**); the squares indicate the fitness of the ‘physical’ robot for each of the three subsequent hardware trials (the behavior 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 center of mass over time (the trajectories are displaced upwards for the sake of clarity).

when the proposed algorithm is used to restore function (the leftmost bars in Figure 6a). However the proposed algorithm only requires three hardware evaluations (more than two orders of magnitude fewer hardware trials) to restore function.

Figure 5 shows that for three sample damage scenarios, much functionality is restored to the ‘physical’ robot after only three hardware trials. In the case of sensor failure for the quadrupedal robot, the forward displacement 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 hops after recovery (note the discrete arcs in the trajectory of its center of mass (Figure 5d)) compared to a more stable but erratic gait before the failure occurred (Figure 5b).

Figure 6a indicates that functionality is restored for all but the severe, compound damage scenarios (scenarios 7 and 8) for the quadruped, and for all but three of the scenarios for the hexapod (scenarios 4, 7 and 10). In these scenarios, the algorithm never converged on the correct damage hypothesis. (This is to be expected for scenario 10, in which it is impossible for the estimation EA to converge on the correct hypothesis, but only to approximate it.) The increased stability of the hexapod allows it to recover from the one of the severe compound damage scenarios (scenario 8).

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 behavior against the physical robot’s behavior, 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 scenarios 1 and 2 (morphological damage) and

scenarios 3 and 5 (controller damage), for both robots, approaches or exceeds original performance. Because the algorithm evolves the robot simulator itself based on the ‘physical’ robot’s experience, it would be straightforward to generalize this algorithm beyond internal damage: the estimation 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 (scenario 10). This type of failure cannot be encoded by the estimation EA, but it is able to approximately describe the failure using aggregates of other damage types: for this run of the algorithm, the failure of a touch sensor (T_1) and the jamming of one of its lower leg joints (M_3) approximates the failure (Figure 5e).

Table II indicates a subset of the synaptic weights for the original evolved controller for this particular case. 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 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 automated 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

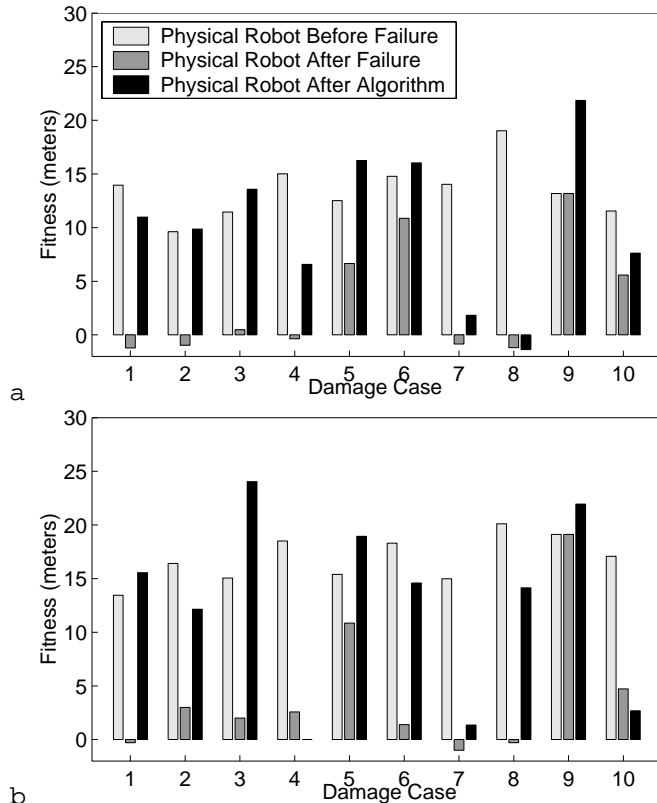


Fig. 6. Results for all of the damage scenarios. **a:** The recovery of the quadrupedal robot for all 10 damage scenarios. The light gray bars indicate the forward displacement of the physical robot before failure; the medium gray bars indicate displacement after failure; and the black bars indicate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

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 periodically provides neural controllers for the 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 change. 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 using aggregates of anticipated damage possibilities.

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 estimation EA can more easily generate damage hypotheses. Also, environmental change will be incorporated into the estimation EA so that the robot can not only recover from internal damage, but respond to environmental change. Finally, we plan to generalize this algorithm to other problem

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					

domains beyond robotics.

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

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.

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-nonsolf 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] 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] JPL Mars Exploration Rovers (2004) <http://www.jpl.nasa.gov/mer2004/>.
- [14] opende.sourceforge.net
- [15] 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.
- [16] K. Sims. Evolving 3D morphology and behavior 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.
- [19] 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.