

DETC2004-57574

## AN EXPLORATION-ESTIMATION ALGORITHM FOR SYNTHESIS AND ANALYSIS OF ENGINEERING SYSTEMS USING MINIMAL PHYSICAL TESTING

Hod Lipson

Josh Bongard

Computational Synthesis Laboratory, Sibley School of Mechanical & Aerospace Engineering  
Cornell University, Ithaca NY 14853, USA

### ABSTRACT

We describe a new general algorithm for the automated design, analysis and repair of nonlinear physical systems. The process iterates a two-phase exploration-estimation cycle. The exploratory phase seeks a new improvement or test to perform to the system based on some initial internal model. The estimation phase performs the suggested operation and observes the outcomes; it then improves the internal model so as to explain all observations so far. This process relies on very few, targeted, and carefully planned interactions with the physical systems. We describe an implementation of this method using two evolutionary algorithms, where the exploratory phase uses a simulator to evolve improvements or tests, and the estimation phase uses observations to evolve the simulator itself. We demonstrate this algorithm for analysis, design and repair of electromechanical systems.

### INTRODUCTION

A common challenge of many synthesis and analysis problems is the need to produce a good solution while doing a minimal number of physical trials. In many *design* problems, an engineer must plan a solution based on some predictive abstraction of the problem, and only then implement and try the solution in reality. If the solution succeeds, the goal has been achieved; but if the solution is unsatisfactory, additional design iterations are required. Iterations involving a physical test are often costly and slow, and great effort is made to reduce the number of iterations by using increasingly higher-fidelity abstractions.

Conversely, in many *analysis* problems, an analyst tries to reverse-engineer a physical system by creating a model compatible with some observation of its behavior. The model is then used to make predictions about the system, and these predictions are then tested. If the predictions are confirmed, the

goal has been achieved; but if the predictions are incorrect, additional analysis iterations are needed. Each iteration involving testing of the physical system is often costly, slow, and possibly disruptive to the analyzed system, and great effort is made to reduce the number of physical interactions by using carefully planned and targeted tests.

Examples for both these cases are abundant: The recent difficulties with JPL's Mars rovers is a dramatic example; both robots suffered different, unanticipated partial failures [1]. Automatic recovery involving both analysis and redesign is most acute in such instances, where human operators cannot manually repair or provide compensation for damage or failure. Similarly, excessive testing of many complex systems, especially those involving mechanical components, is costly, slow and may lead to wear and degradation.

Human engineers reduce the need for physical experimentation ('trial and error') by creating an *abstraction* of the physical system. This abstraction – be it an analytical model, a computational simulation, or a mental insight – is used to predict properties and capture knowledge about the system. Only design improvements that are expected to be the most successful, or tests that are predicted to yield the most relevant new information, are actually performed in reality. It is this kind of abstraction that must be a component of any effective design automation algorithm that deals with a physical system.

The analysis and synthesis problems are not independent: For effective *design*, also an analysis and understanding of a system is required; for effective *analysis*, also a careful design of experiments is required; and for the *repair* of a system, both analysis for diagnostic and redesign for recovery are required. It is thus not surprising that an algorithm that performs either of these functions will have to do both, and this is what we present here.

## THE EXPLORATION-ESTIMATION ALGORITHM

The algorithm we present iterates a two-phase exploration-estimation cycle. The exploratory phase seeks a new design improvement or test to perform on the system based on some initial internal model. The estimation phase performs the suggested operation and observes the outcomes; it then improves the internal model so as to explain all observations so far. This algorithm can be viewed as an antagonistic **co-evolution** of models and tests, where models are evolved to **explain** tests, and tests are evolved to create **disagreement** among model predictions.

### Terminology

A **system** is defined as a ‘box’ with **inputs** and **outputs**. We assume the system is typically nonlinear and holds internal state. For example, the system may be a finite state machine or a physical machine. The system may include a robot, its environment *and* its controller; in this case the inputs are system specifications, and the output is the observable behavior of the robot. Note that this is different from the conventional control-theoretic view that separates the system from the control. Here the system includes both the machine and control, and the design algorithm is outside the system, trying to provide inputs to modify the system (the morphology and/or the control) so as to get the target behavior (for synthesis) or learn about the system (analysis).

A definition of a system must include a **representation** language to represent the space of systems, the space of inputs, and the space of outputs, and **operators** to search these spaces. We also need a **distance metric** to compare inputs to inputs, outputs to outputs, and systems to systems. The metric should be zero for identical arguments and positive for non-identical

arguments, and should obey the triangle inequality. The algorithm uses multiple system models, and each is referred to as a hypothesis. For design or repair, an external **target output** should also be provided.

The following components are required:

- A representation, search operators and similarity metric for systems
- A representation and search operators for inputs
- A similarity metric for outputs
- For design or repair, a target output

### Algorithm outline

1. **Initialization:**
  - a. Define a set of initial models (hypotheses) of the system. They can be blank, random, or seeded with some prior knowledge.
2. **Exploration phase**
  - a. Search for an input that creates the most variation among the outputs of the current set of model hypotheses.
  - b. For design or repair, search for the input that yields the output that is closest to the target output on the current set of model hypotheses.

*Note:* For design/repair, the above two goals must be alternated, staggered, or combined by weighting or some other multi-objective search technique.
3. **Estimation phase**
  - a. Test the physical system by applying the inputs and measuring the outputs. This input-output set is stored.
  - b. Search the model space for models that are maximally consistent with *all* stored input-output sets. Multiple,

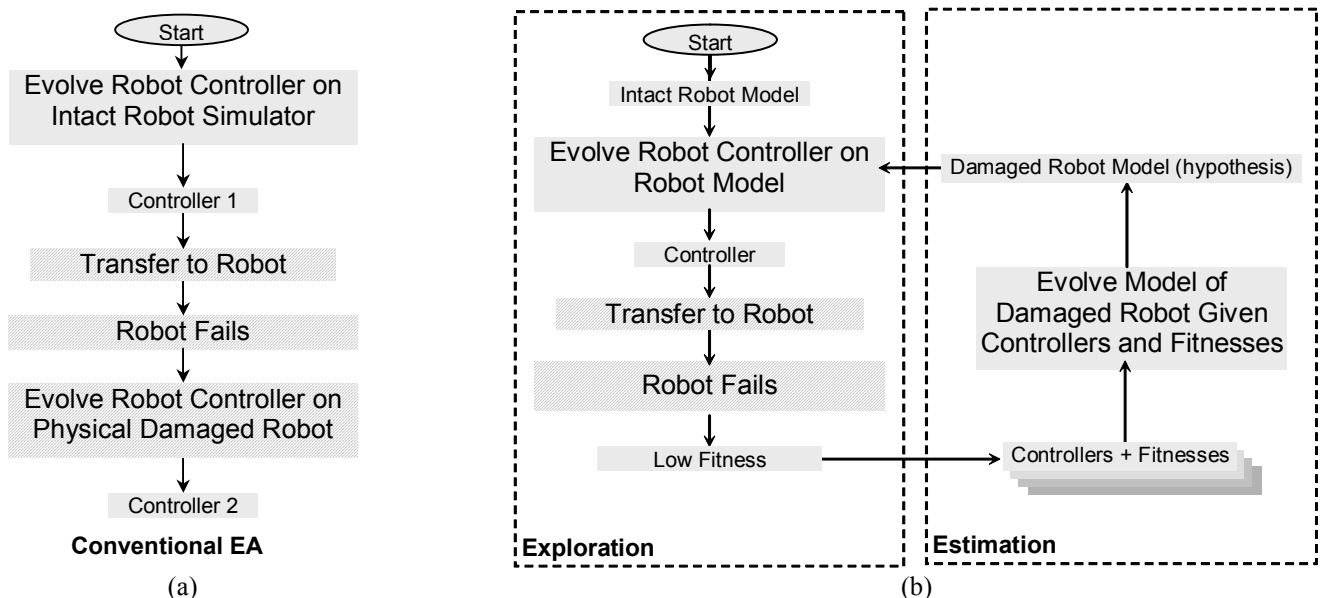


Figure 1. Exploration-estimation Evolutionary Algorithm (EA) for recovering functionality. (a) A conventional EA recovers functionality by extensive testing on the physical robot (b) Two-phase exploration-estimation EA. Steps that take place on the physical robot are crosshatched.

different candidate hypotheses are required.

#### 4. Termination

The exploration-estimation cycles are repeated until one of these conditions is met:

- a. The estimation phase is unable to produce a model consistent with all the accumulated observations. In this case the algorithm failed, but at least avoids unnecessary blind trials
- b. For synthesis: The target output has been achieved on the physical system.
- c. For analysis: The estimation phase repeatedly produces the same set of model hypotheses for a number of cycles in a row, or the exploration phase is unable to find an input that causes variation in the outputs of the current model hypotheses. The hypotheses may be identical, indicating there is one solution; or different, indicating that some aspect of the system is unobservable and several solutions are possible but cannot be disambiguated by the inputs.

#### 5. Cross validation (for analysis)

Upon successful termination in analysis (4.c), it is necessary to validate that the system was reliably inferred. This may be done by generating random or arbitrary inputs and testing the outputs against prediction of the models. Successful predictions provide a positive confirmation; unsuccessful predictions should be included in the test data and the estimation-exploration cycle resumed.

#### Modes of failure or non-convergence

If the algorithm fails, there are a number of possible causes:

- The *physical* test was not carried out properly (the inputs were not set correctly or the measurements of the outputs were incorrect)
- The search process in the estimation phase is not able to find the best models (cause depends on the search technique used, e.g. could be stuck in local minima)
- The system or necessary inputs are outside the space spanned by the representation language.
- The system behaves inconsistently. The space of systems may be broadened to include additional, possibly non-deterministic elements.

If the algorithm continues running but does seem to make progress (i.e., is slow to converge), there are a number of possible causes

- The search process in the estimation phase does not produce a diverse enough set of hypothesis (lack of diversity). In this case the inputs generated in the exploration stage do not produce much information beyond what is already known and the algorithm cannot make progress.
- The search process in the exploration phase is not able to find the best inputs (cause depends on the search technique used)

#### Implementation using evolutionary computation

The exploration-estimation algorithm can be implemented using a variety of search algorithms. Particularly appealing is the use of evolutionary search techniques as they are inherently population-based and can produce the diversity of solutions required for this algorithm. However, any other search technique can be used as long as it is capable of producing multiple satisfactory solutions for the problem domain. For example, a parallel hill-climber, parallel simulated-annealer, or heuristic search like A\* can be used.

Figure 1 shows an example of applying the exploration-estimation algorithm to the design and analysis of a robot controller, where the physical robot's morphology, environment, or damage (if any) are unknown.

#### **APPLICATION TO ROBOTICS**

We have applied the presented algorithm to the automated design, analysis and repair of robotic systems. We focus here on evolutionary robotics, a design automation technique that has been used for the open-ended design of robot controllers [7,9] and morphologies [10,8,11,12,13].

Many of the current evolutionary design algorithms assume an accurate simulator exists for the design space being searched. However this raises a major challenge concerning the transferal of evolved machines from simulation to reality, or 'crossing the reality gap' [2]. Alternatively, some evolutionary design processes work by directly modifying the physical system [3,4,5] or evolving first in simulation followed by further adaptation in reality [6]. This approach requires extensive amount of physical testing (on the order of 1,000-10,000 trials) that is prohibitive for most physical systems, especially those involving mechanical components.

The following sections describe a number of experiments in which various instantiations of the presented algorithm have been applied for the analysis, redesign and repair of a legged robot. In these experiments the 'physical' robot is also simulated, but with different parameters, failures and environmental conditions unknown to the internal simulator. We first describe the robotic platform itself, and then provide details about each of the experiments and results.

#### The Robot

In this work a quadrupedal robot was simulated and used to test the algorithm (Figure 2). The simulator works in unison with a 'physical' robot, which in this case is also simulated [14]. 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 2a. The robot also has four binary touch sensors, and four angle sensors that return values in  $[-1,1]$  commensurate with the angle of the joint to which they are attached. The robot is 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 2b. 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 threshold activation function is applied at the neurons.

### Application of the algorithm to robot analysis

The objective of this experiment is to recover the physical properties of the physical robot morphology, using minimum number of physical experiments. This is similar to a system-identification task, but applied to a highly nonlinear system.

We implemented the exploration-estimation algorithm with the following specific implementation choices:

1. **System:** The quadrupedal robot, including both morphology and control as shown in Figure 2. **System hypothesis representation:** The genomes of the estimation phase encoded some aspects of the simulated robot's morphology that are thought to differ between the simulated and physical robot. In this experiment they encoded the masses of the robot's body parts (in kilograms) and sensor time lags (in simulator time steps). **System search operators:** Conventional mutation and crossover operators of evolutionary search. **System similarity metric:** The length of time that the sensor signals from the simulated and target robot stay correlated: behaviors that diverge early indicate a poorer simulation compared to signals that stay correlated longer (for a more detailed description of this metric, see [15,17,18]).
2. **Inputs:** All synaptic weights of the neural controller. **Inputs search operators:** Conventional mutation and crossover operators of evolutionary search.
3. **Outputs:** The behavior of the robot, specifically the time series produced by its sensors. **Outputs similarity metric:** Hamming distance between corresponding sensory signals during the first 20 time steps of behavior. Since the system is highly nonlinear, even very similar behaviors diverge rapidly and more elaborate metrics need to be used for longer-term comparisons. **Target outputs:** not needed here, since this is an analysis experiment.

We used a standard genetic algorithm to perform both the exploration and estimation phases, with a population size of 100 and 30 generations per cycle. In the estimation stage, system hypotheses were evolved. In the exploratory phase, new tests were evolved. We compare this to a control experiment

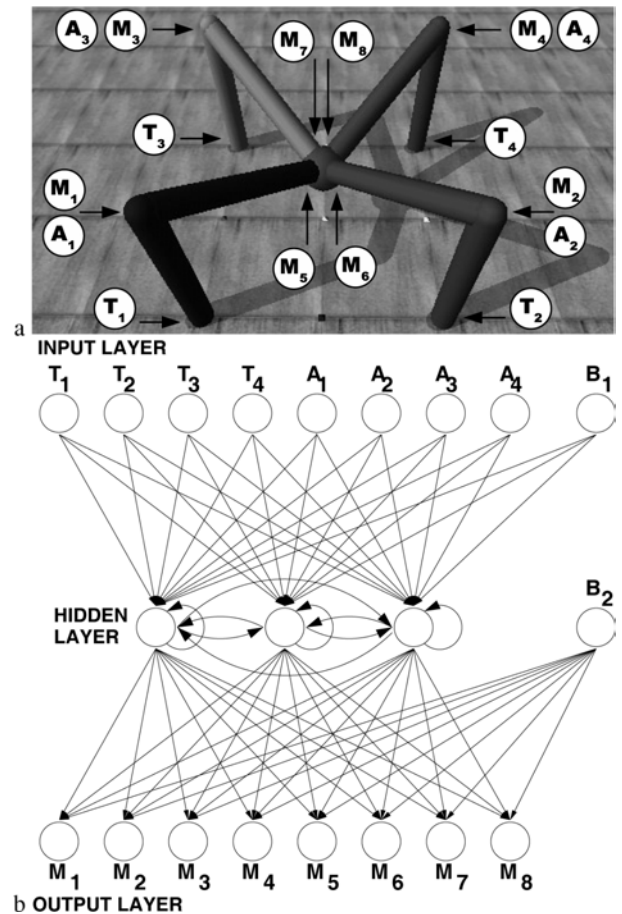


Figure 2. **The quadrupedal test robot:** (a) The morphology of the robot, including the distribution of its four touch sensors (T1-T4), four angle sensors (A1-A4), and eight motorized joints (M1-M8), and (b) The neural network controller of the robot, which connects the eight sensors to the eight motors via a single hidden layer, and an additional two bias neurons (B1-B2).

that used feedback from a single test but performed the same number of evaluations.

**Results** Fifty independent runs of the algorithm were conducted against the target robot outlined in Table I. Figure 3 shows the 50 series of 20 best simulator modifications output after each pass through the estimation phase.

**Table I: Differences between the simulated and target robots**

Body Part or Sensor	Simulated Body Part	Target Body Part	Simulated Sensor	Target Sensor
1	5.0 (kg)	6.5 (kg)	0 (t)	15 (t)
2	1.0	2.4	0	7
3	1.0	2.0	0	19
4	1.0	3.0	0	4
5	1.0	2.7	0	10
6	1.0	1.9	0	5
7	1.0	2.8	0	18
8	1.0	2.7	0	16
9	1.0	2.4		

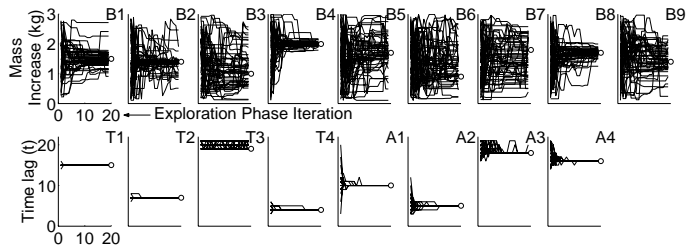


Figure 3. **The repeated convergence during the 50 independent runs to the correct morphological properties of the target robot.** The circles indicate the specific morphological difference between the default simulated robot and the target robot (see Table I). The top row indicates the convergence of the algorithm toward the actual mass distribution of the target robot. The bottom row indicates the convergence toward the actual time lags of the eight sensors.

One of the runs was selected at random, and the gait of the simulated robot was compared against the gait of the target robot, when both used the same evolved controller. Figure 4a indicates the change in behaviors when the first evolved controller was transferred, and Figure 4b shows the behavior change when the 20<sup>th</sup> evolved controller was transferred, during the last iteration through the algorithm's cycle. Finally, for each pass through the exploration phase, the distance traveled by the simulated robot was averaged over all the 50 runs. Similarly, the distance traveled by the target robot using the same controller was averaged over the 50 runs. The results are shown in Figure 5.

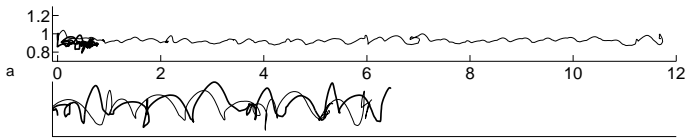


Figure 4. **Behavior recovery after controller transferal.** After the first pass through the exploration phase, the best evolved controller was used by the default simulated robot. The trajectory of its center of mass is given by the thin line in a. The same controller was then supplied to the target robot, and the resulting trajectory of its motion is given by the thick line in a. The movement of the simulated robot in the updated simulator after the 20th pass through the exploration phase (using the new best evolved controller) is given by the thin line in b. The motion of the target robot using the same controller is given by the thick line in b. The horizontal axis indicates forward distance, and the vertical axis indicates height (both are in meters).

Figure 3 makes clear that for all 50 runs, the algorithm was better able to infer the time lags of the eight sensors than the mass increases of the nine body parts. This is not surprising in that the sensors themselves provide feedback about the robot. In other words, the algorithm automatically, and after only a few target trials, deduces the correct time lags of the target robot's sensors, but is less successful at indirectly inferring the masses of the body parts using the sensor data.

Convergence toward the correct mass distribution, especially for body parts 1, 2, 4 and 8 can be observed. Even with an approximate description of the robot's mass

distribution, the simulator is improved enough to allow smooth transfer of controllers from simulation to the target robot. Using the default, approximate simulation, there is a complete failure of transferal, as indicated by Figure 4a: the target robot simply moves randomly, and achieves no appreciable forward locomotion.

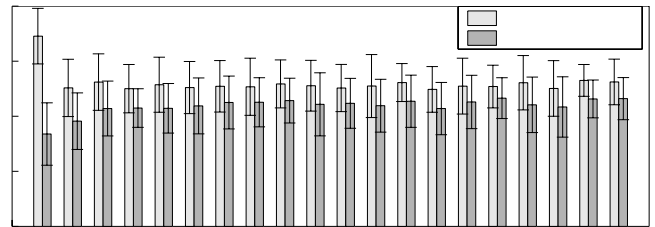


Figure 5. **Average transferal success after each target trial.** The light gray bars indicate the average distance traveled by the simulated robot using the best evolved controller output by that pass through the exploration phase, over all 50 runs. The dark gray bars indicate the average distance traveled by the target robot using the same controller, during each target trial. Error bars indicate two units of standard deviation.

After 20 iterations through the algorithm, an improved simulator is available to the exploration phase, which evolves a controller that allows the simulated robot to move forward, although not as far as the original simulated robot (indicated by the shorter trajectory in Figure 4b compared to Figure 4a). Also, the new gait causes the robot to hop (indicated by the large vertical curves of the robot's center of mass in Figure 4b) instead of walk (indicated by the steady trajectory of Figure 4a). In contrast to the first pass, the target robot exhibits very similar behavior to the simulated robot when it uses the same controller: both travel a similar distance (about 6.5m), and both move in the same way (both exhibit a hopping gait that produces trajectories with similar frequencies and amplitudes).

Finally, Figure 5 shows that this improvement in behavior transferal success is a general phenomenon. On average, over the 50 independent runs, there is a drop by 50% in the distance traveled by the target robot, compared to the default simulated robot. After about five iterations through the algorithm's cycle there is only a statistically insignificant decrease in distance traveled between the two robots. Although not shown in Figure 5, this similar distance is matched in all of the cases viewed by a qualitative similarity in gait patterns, as shown for a single run in Figure 4b.

It is useful to stress that the algorithm discovers the correct mass distribution without any direct access to information about the target robot's morphology: the morphology is determined based solely on returned sensor data. Figure 4 shows that the algorithm is better at guessing the masses of some body parts than others. However even without sensors attached directly to the all parts of the body, the algorithm is able to guess the morphological properties correctly in many cases.

### Application to robot repair

The objective of this experiment is to restore operation to a damaged robot. There is no direct information about the damage that may have occurred, but the functionality must be restored by redesigning a controller that is *qualitatively* different than the original controller.

In order to recover functionality for a damaged robot, the exploration-estimation algorithm was used to model the damage suffered by a physical robot (which is also simulated in our current work). Three hundred independent runs were executed, in which the 'physical' robot underwent various types of damage: failure or sensor or motors; separation of body parts; changes in mass distribution; actuated joints become jammed; or there is some change in the robot's environment, such as a horizontal tilting of the ground plane. Thirty independent runs for each of 10 different failure scenarios (listed in Table II) were executed.

Table II: Unanticipated scenario cases

Case	Explanation
1	One motor weakens by 50%.
2	One body part increases in mass by 200%
3	One of 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 robots 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%.

The exploratory phase was identical as for that described for the robot analysis application; input was a set of synaptic weights for the neural network controller. Fitness---and therefore a good test---was determined as the maximum forward distance traveled by the target robot. The system hypothesis representation is here a set of morphological, neural or environmental modifications that are applied to the simulation. Fitness in this case is how well the exploratory phase 'breaks' the robot, or modifies the environment, so as to approximate the crippled state of the 'physical' robot or its novel environment. For each of the 30 independent runs, a unique set of robot failures or environmental changes were instituted; each phase was cycled through 30 times, requiring only 30 target trials. Within each phase, an evolutionary algorithm containing a population of either 100 sets of synaptic weights, or 100 sets of system hypotheses, was run for 30 generations.

The progress of a typical run is shown in Figure 6, where function recovery is achieved after a correct diagnosis of the failure (partial sensor failure) is evolved. The average function recovery of the robot for these 10 scenarios, plus three scenarios that can not be perfectly approximated by the

estimation phase, are shown in Figure 7. As can be seen, in almost all cases, function is restored to the crippled robot.

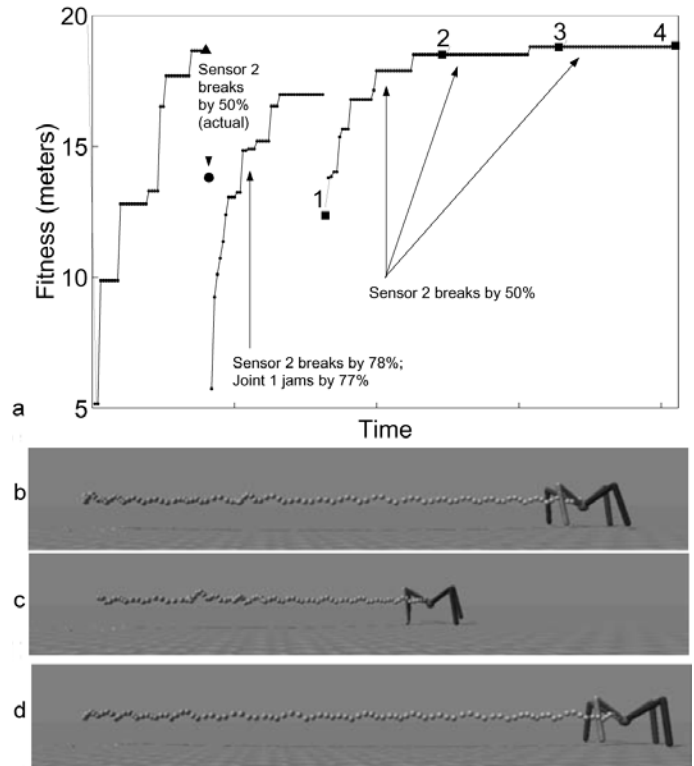


Figure 6. The repair progress of for unanticipated situation number 5: one of the angle sensors breaks by 50%. (a) The stepped lines indicate the progress of the five passes of the exploration phase. The captions indicate the best hypotheses evolved by the four passes through the estimation phase. The triangle shows the original fitness (distance traveled in meters) of the physical robot. The circle indicates distance traveled after experiencing the unanticipated situation. The squares indicate the distance traveled by the recovered physical robot during each of the four hardware trials. (b-d): Trajectories of the physical robot's motion: (b) before the unanticipated situation; (c) after encountering the unanticipated situation; and (d) during the fourth hardware trial.

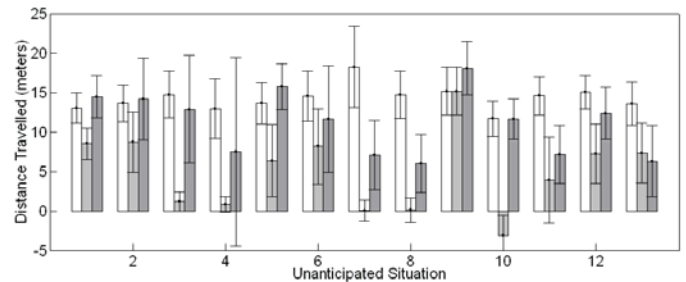


Figure 7. The average performance of the presented algorithm for all unanticipated situations listed in Table I. The white bars indicate the average distance traveled by the 'physical' robot before encountering the unanticipated situation; the light gray bars after it has encountered the situation; and the dark gray bars indicate the distance traveled by the 'physical' robot during the fourth hardware trial.

## CONCLUSIONS

We have described a new general algorithm for the automated design, analysis and repair of nonlinear physical systems. The process iterates a two-phase, co-evolutionary estimation-exploration cycle. The exploratory phase seeks a new improvement or test to perform to the system based on some initial internal model. The estimation phase performs the suggested operation and observes the outcomes; it then improves the internal model so as to explain all observations so far.

The key advantage of this algorithm is when design, analysis or repair must be performed to a physical system under uncertainty – where some aspects of the system are unknown but extensive physical testing is prohibitive and extensive data about the system is not available. In the robot repair application, we showed how a complex, nonlinear system subject to a compound damage was redesigned to restore functionality using only *four* physical tests – this stands in contrast to other approaches that need explicit feedback from failure sensors, or perform large amounts of tests on the physical systems.

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 ([16]), and reverse engineering of large finite state machines ([19]). Future avenues of study will include applying our robot-based algorithm to an actual physical robot, as well as generalizing the algorithm to a wider range of coupled, non-linear systems.

## ACKNOWLEDGMENTS

This work was supported by the U.S. Department of Energy, grant DE-FG02-01ER45902.

## REFERENCES

1. JPL Mars Exploration Rovers (2004) <http://www.jpl.nasa.gov/mer2004/>.
2. Jakobi, N. (1997). Evolutionary robotics and the radical envelope of noise hypothesis. *Adaptive Behavior*, 6(1):131–174.
3. Thompson, A. (1997). Artificial evolution in the physical world. In Gomi, T., editor, *Evolutionary Robotics: From intelligent robots to artificial life (ER'97)*, pages 101–125. AAI Books.
4. Floreano, D. and Mondada, F. (1998). Evolutionary neurocontrollers for autonomous mobile robots. *Neural Networks*, 11:1461–1478.
5. Mahdavi, S. H. and Bentley, P. J. (2003). An evolutionary approach to damage recovery of robot motion with muscles. In *Seventh European Conference on Artificial Life (ECAL03)*, pages 248–255. Springer.
6. Pollack, J. B., Lipson, H., Ficici, S., Funes, P., Hornby, G., and Watson, R. (2000). Evolutionary techniques in physical robotics. In Miller, J., editor, *Evolvable Systems: from biology to hardware*, pages 175–186. Springer-Verlag.
7. Nolfi S., Floreano D. (2000), *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*, MIT Press
8. Lipson, H. and Pollack, J. B. (2000). Automatic design and manufacture of artificial lifeforms. *Nature*, 406:974–978.
9. Reil, T. and Husbands, P. (2002). Evolution of central pattern generators for bipedal walking in a real-time physics environment. *IEEE Transactions on Evolutionary Computation*, 6(2):159–168.
10. Sims, K. (1994). Evolving 3D morphology and behavior by competition. *Artificial Life IV*, pages 28–39.
11. Hornby G.S., Lipson H., Pollack. J.B., 2003 “Generative Encodings for the Automated Design of Modular Physical Robots”, *IEEE Transactions on Robotics and Automation*, Vol. 19 No. 4, pp 703-719
12. Bongard, J. and Pfeifer, R. (2001). Repeated structure and dissociation of genotypic and phenotypic complexity in Artificial Ontogeny. *Proceedings of The Genetic and Evolutionary Computation Conference (GECCO 2001)*, pages 829–836.
13. Adamatzky, A., Komosinski, M., and Ulatowski, S. (2000). Software review: Framsticks. *Kybernetes: The International Journal of Systems & Cybernetics*, 29:1344–1351.
14. Simulations were based on Open Dynamics Engine, an open-source 3D dynamics simulation package.
15. Bongard, J. C. and Lipson, H. (2004a). Automated robot function recovery after unanticipated failure or environmental change using a minimum of hardware trials. In *Procs. of the 2004 NASA/DoD Conference on Evolvable Hardware*.
16. Bongard, J. C. and Lipson, H. (2004b). Automating genetic network inference with minimal physical experimentation using coevolution. In *Procs. of the Genetic and Evolutionary Computation Conference (GECCO 2004)*.
17. Bongard J., Lipson H. (2004), “Automated Damage Diagnosis and Recovery for Remote Robotics”, *IEEE International Conference on Robotics and Automation (ICRA04)*, pp. 3545-3550
18. Josh C. Bongard and Hod Lipson, (2004) “Once More Unto the Breach: Automated Tuning of Robot Simulation using an Inverse Evolutionary Algorithm”, *Ninth Int. Conference on Artificial Life (ALIFE IX)*, to appear
19. Josh C. Bongard and Hod Lipson, (2004) “Topological system identification using coevolution of models and tests”, in review.