Once More Unto the Breach[1]: Co-evolving a robot and its simulator

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

Abstract

One of the major challenges facing evolutionary robotics is crossing the reality gap: How to transfer evolved controllers from simulated robots to real robots while maintaining the behavior observed in simulation. Most attempts to cross the reality gap have either applied massive amounts of noise to the simulation, or conducted most or all of the evolution onboard the physical robot, an approach that can be prohibitively costly or slow. In this paper we present a new co-evolutionary approach, which we call the *estimation-exploration algorithm*. The algorithm automatically adapts the robot simulator using behavior of the target robot, and adapts the behavior of the robot using the robot simulator. This approach has four benefits: the process of simulator and controller evolution is automatic; it requires a minimum of hardware trials on the target robot; it could be used in conjunction with other approaches to automated behavior transferal from simulation to reality; and the algorithm itself is generalizable to other problem domains. Using this approach we demonstrate a reduction of three orders of magnitude in the number of evaluations on a target robot (thousands compared to only five).

Introduction

An evolutionary robotics experiment requires an evolutionary algorithm to optimize some aspect of a simulated or physical robot in order to generate some desired behavior. Because it is difficult and very time-consuming to perform the thousands of fitness evaluations required by an evolutionary algorithm on a physical robot, most or all of evolutionary robotics experiments are performed in simulation.

Evolution in simulation raises a major challenge: The transferal of evolved controllers from simulated to physical robots, or 'crossing the reality gap' (Jakobi, 1997). There are several approaches to this challenge, including: adding noise to the simulated robot's sensors (Jakobi, 1997); adding generic safety margins to the simulated objects comprising the physical system (Funes and Pollack, 1999); evolving directly on the physical system ((Thompson, 1997), (Floreano and Mondada, 1998) and (Mahdavi and Bentley, 2003)); evolving first in simulation followed by further adaptation on the physical robot ((Pollack et al., 2000) , (Mahdavi and Bentley, 2003)); or implementing some neural plasticity that allows the physical robot to adapt during its lifetime to novel environments ((Floreano and Urzelai, 2001), (DiPaolo, 2000), (Tokura et al., 2001)).

Another approach that can be used in lieu of, or in addition to the above-mentioned approaches is outlined in this paper. Our approach uses a co-evolutionary algorithm to automatically evolve a robot simulator so that the simulated robot acts more like the target robot. Instead of only evolving a controller in simulation and transferring it to the target robot, sensory data recorded by the target robot is returned to the algorithm and used to evolve the *morphological* aspects of the simulated robot. This approach has two benefits: it automates the process of adapting a simulation to a particular physical system; and it accomplishes this with a minimum of hardware trials.

Most evolutionary robotics experiments evolve controller parameters for a robot with a fixed controller topology and fixed morphology (examples include (Floreano and Mondada, 1998) and (Reil and Husbands, 2002)), whether the evolution is performed in simulation or the real world. Other approaches have widened evolution's control over the design process by subjugating the controller topology and/or the robot's morphology to modification as well (eg. (Sims, 1994), (Adamatzky et al., 2000), (Lipson and Pollack, 2000), (Bongard and Pfeifer, 2001) and (Hornby and Pollack, 2002)) with the aid of simulation.

The ability to evolve a simulated robot's morphology, in addition to its controller, has become much easier recently due to the advent and availability of physics-based simulators, which allow for faster than real-time evaluation of different physical systems. In the next section we present our algorithm, which relies on the physical simulator to update the morphology of the simulated robot to better approximate the morphology of a 'physical' robot. In the work presented here, the 'physical' robot is also simulated, but is different than the simulated robot in ways unknown to the algorithm. In future work we plan to replace the simulated target robot with a physical target robot. The section after that presents some results; the penultimate section provides some discussion regarding the generality and applicability of our approach for future physical evolutionary robotics ex-

[1] Shakespeare, W. *King Henry V*, Act III, Scene 1. **Breach:** [n] an opening (especially a gap in a dike or fortification). Also [n] a failure to perform some promised act or obligation.

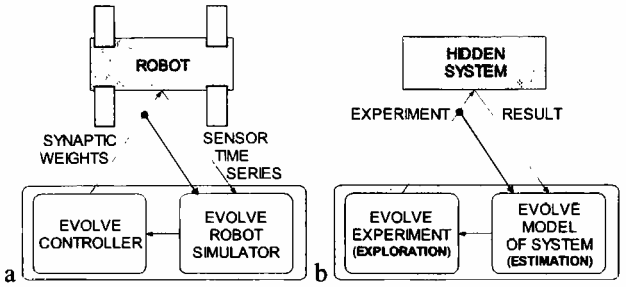

Figure 1: **Schematic view of the proposed algorithm. a:** The cyclical flow of the estimation-exploration algorithm, as applied to evolutionary robotics. The exploration phase evolves controllers in simulation and transfers them to a target robot; based on sensor time series and the evolved controller, the estimation phase improves the simulation. **b:** The general layout of the algorithm for inferring the structure of some hidden target system.

periments; and the final section offers some concluding remarks.

Methods

The algorithm presented in this paper takes as input a target robot, and an initial approximate simulator of the target robot. In other words, the initial simulated robot is morphologically different, in some unknown way, from the target robot. After the algorithm is completed, two structures are returned. The first is a robot simulator that better describes the target robot. The second is an evolved controller that should produce behavior in the target robot that is very similar to the behavior that was observed in the robot simulator, using the same controller. Figure 1 shows the algorithm flow schematically, as well as the flow of the algorithm for any partially hidden, nonlinear system.

The Algorithm. The estimation-exploration algorithm is comprised of two phases: the exploration phase, which in this case evolves neural network controllers; and the estimation phase, which in this case evolves improvements to the initial approximate robot simulator, given pairs of evolved controllers and resulting sensor data obtained from the target robot. The algorithm is cyclical. First, using the default simulator, the exploration phase evolves a controller that allows the robot to move as far forward in its environment as possible, during a fixed time period. Then, the best evolved controller is applied to the target robot, and the sensor values over time are recorded.

Using this evolved controller/sensor data pair (plus any previously evolved controller/sensor data pairs obtained during previous cycles), the estimation phase evolves simulator modifications that allow the simulated robot to better reproduce the observed sensor data, given the evolved controller. The best evolved simulator (that which best reproduces the i behaviors observed during the previous i passes through the algorithm) is then passed to the exploration phase, and the cycle continues for a number of times. A more detailed explanation of the algorithm is given after the robot simulator and the robot is itself are described.

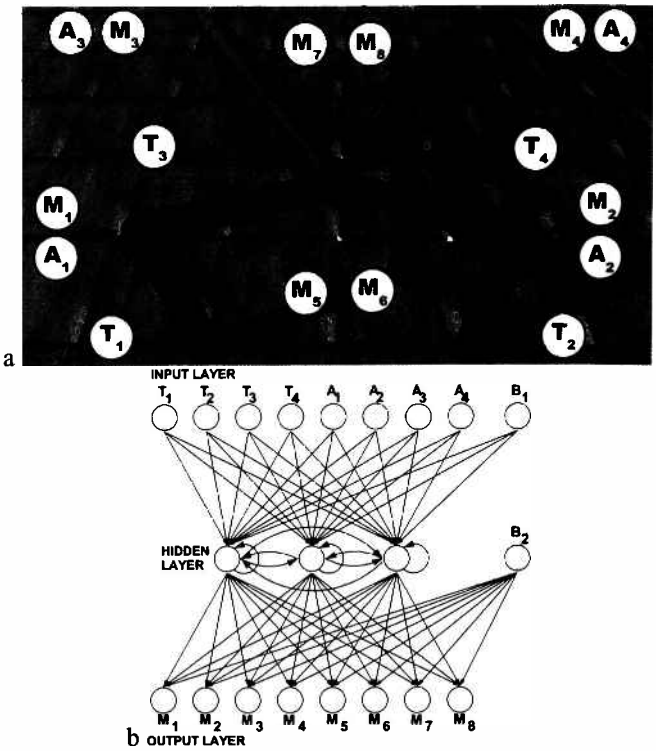

Figure 2: **a:** The morphology of the robot, including the distribution of its four touch sensors (T1-T4), four angle sensors (A1-A4), and eight motorized joints (M1-M8). **b:** The neural network controller of the robot, which connects the eight sensors to the eight motors via a single hidden layer,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

The Robot. In this work a quadrupedal robot was simulated and used to test the algorithm. The robot simulator is based on Open Dynamics Engine, an open-source 3D dynamics simulation package[1]. 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 away from their starting angles. 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. In this work the target robot is not a physical robot out in the world, but a separate, simulated robot which is identical to the default simulated robot except for some unknown morphological differences. The task of the algorithm is to indirectly infer these differences, and modify the default simulator to reflect them accurately.

Both robots (simulated and target) are controlled by a partially recurrent neural network, which receives sensor data at the beginning of each time step of the simulation into its input layer, propagates those signals to a hidden layer containing three hidden neurons, and finally propagates the signals to an output layer. The neural network architecture is shown in Figure 2b. Two types of sensors are used: touch

[1] `opende.sourceforge.net`

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. At each time step the values arriving at the eight motor neurons are treated as desired joint angles. The difference between the joint's actual angle and the desired angle output by the corresponding motor neuron is computed, and a torque commensurate with this difference is applied to the joint. All neuron values and the 68 synaptic weights are scaled to lie in the range $[-1.00, 1.00]$. A threshold activation function is applied to the hidden and motor neurons. Once the torques have been applied to the joints, the other external forces acting on the robot's body parts (gravity, friction, and inertia) are summed, and the positions, velocities and accelerations are updated at the next time step. Evaluation continues for 1000 time steps.

Algorithm Application. Our algorithm is quite generalizable: to date, we have applied to the algorithm to other system identification problems such as gene network inference (Bongard and Lipson, 2004b) and function recovery for damaged robots (Bongard and Lipson, 2004a), and other problem domain applications are planned. In order to apply the algorithm to a particular problem (which in this paper is the automated evolution of a robot simulator in order to cross the reality gap) four steps must be followed: **initialization**, setting up the **exploration phase**, setting up the **estimation phase**, and finally setting the **termination criteria**.

1) Initialization: The algorithm begins with the default robot simulator described above, and starts to evolve a controller in the exploration phase.

2) Exploration Phase: The exploration phase uses a given robot simulator to evolve a set of synaptic weights that, when applied to the neural network shown in Figure 2b, causes the robot to move forward as far as possible in a fixed time period. During the first pass through the exploration phase, the default simulator is used; in subsequent passes through this phase, the best evolved simulator from the estimation phase is used.

Genomes in the exploration phase are strings of 68 floating-point values in $[-1, 1]$, representing the 68 synaptic weights of the network. The fitness of a given genome is simply the z component (forward distance) of the robot's center of mass (in meters) during the first time step of the simulation subtracted from the z component of the center of mass at $t = 1000$.

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. Each floating-point value of a copied genome has a 1% chance of undergoing a point mutation. Each value selected for mutation is either: nudged up or down by 0.0001 (50% probability); or replaced with a new random value in $[-1, 1]$. Of the 100 copied and mutated genomes, 24 pairs are randomly selected and undergo one-point crossover. The population is evolved for 30 generations.

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		

Table 1: The set of differences between the default simulated robot and the target robot. Note that the physical characteristics of the target robot (third and fifth columns) are hidden from the algorithm.

When this phase terminates, the controller with the best fitness is downloaded to the target robot, and the resulting sensor values are recorded. Both the evolved controller and resulting sensor time series are passed into the estimation phase.

3) Estimation Phase: The genomes in the estimation phase encode modifications to be made to the default robot simulator such that the resulting simulated robot mimics as well as possible the observed behavior of the target robot.

The experimenter must choose sets of morphological characteristics that may differ between the simulated and target robot, or those characteristics which greatly affect behavior. For the work here, two such characteristics were chosen: mass distribution and sensor time lags. Each genome then in the estimation phase is a string of 17 values. The first nine values indicate mass changes, in kilograms, to be made to the default masses of the robot's nine body parts. The last eight values correspond to the robot's eight sensors, and indicate, in time steps, how long it takes the recording of a sensor signal to reach the corresponding input neuron. Each genome encodes 17 values in $[-1, 1]$. During parsing the first nine values are scaled to lie in $[0, 3]$, indicating a mass increase in 0 to 3kg for that body part (we assume we know that the target robot is heavier in some way from the default simulated robot). The last eight values are scaled to $[0, 20]$, to indicate that a sensor may have no time lag up to a 20 time step time lag. Table 1 indicates the morphological settings for the robot in the default simulator, and the hidden morphological settings for the target robot.

For these initial experiments, we assume that the target robot only differs from the default simulation in those characteristics that we have chosen to place under evolutionary control. From similar experiments (Bongard and Lipson, 2004a) we have found that in some cases even an approximate simulator that does not refine all the differing physical characteristics between the simulated and target robot allows for adequate transfer of behavior.

The quality of a candidate simulator is given by the ability of the simulated robot to mimic the observed behavior of the

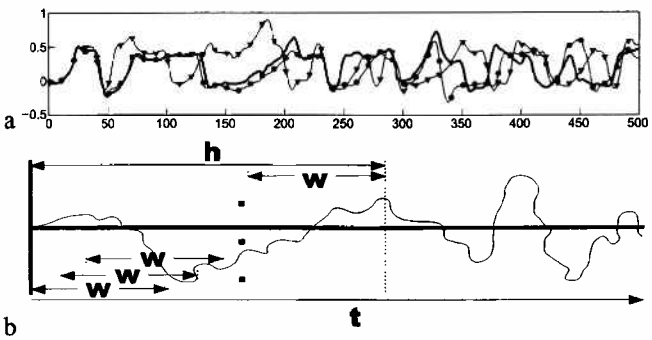

Figure 3: **a**: The three lines represent the time histories of one of the simulated robot's angle sensors: each time history corresponds to three different simulated robots using the same controller. Only the first 500 time steps of the 1000 time step evaluation period are shown. The thick line indicates the angle sensor's values when one of the robot's motors is weakened by 10%; the line with circle markers when the same motor is weakened by 20%; and the line with triangle markers when one of the touch sensors is weakened by 50%. **b:** Outline of the rolling mean fitness metric.

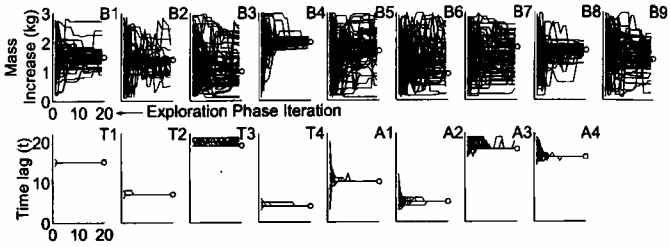

Figure 4: **Convergence toward the physical characteristics of the target robot.** Each pass through the estimation phase produces a set of mass changes for each of the nine body parts of the robot (top row) and a set of time lags for each of the eight sensors (bottom row). Each trajectory represents the best guess output by each pass through the estimation phase of a single run. A total of 50 runs were executed. The open circles indicate the actual differences between the target robot and the starting default simulated robot (for example the first body part of the target robot is 1.5kg heavier than the corresponding body part of the default simulated robot).

target robot. More specifically, given i previously evolved controllers tested on the target robot, the simulated robot should mimic as closely as possible the i sets of sensor time series produced by the target robot. However, quantitatively comparing sensor data from two highly coupled, highly nonlinear machines like the robot used here is very difficult: slight differences between the two machines rapidly leads to uncorrelated signals. To address this, we have formulated a comparison metric called the *rolling mean metric*. This metric has been shown to achieve a high value for morphologically dissimilar robots, a lower value for similar robots, and zero for identical robots (Bongard and Lipson, 2004a).

More specifically, the fitness of a genome g_p in the estimation phase is given by:

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

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

In this formulation i is the number of evolved controllers tested on the target robot so far; j is the number of sensors contained in the robot (here $j = 8$); h is some header length, which indicates how much of initial sensor time series data to use (here $h = 20$); and w indicates the width of a time window within this time header (here $w = 5$). $d(i,j,k,p)$ indicates the differences between the sensor values of the target robot and the simulated robot when it is morphologically modified by the values encoded in genome g_p. $d(i,j,k,p)$ computes a piecewise comparison between short time periods of sensor activation, namely time periods described by $[t-w/2, t+w/2]$. $s_{\text{act}}^{(i,j)}(t)$ indicates the activation of the jth sensor at time t obtained from the target robot when it is using controller i, and $s_{\text{obj}}^{(i,j,p)}(t)$ indicates the activation of the jth sensor at time t from the simulated robot using controller i and modified by genome g_p. We refer to this metric as the rolling mean metric because it compares sets of average sensor activations over a short initial time period. The rolling mean helps smooth the sawtooth activation patterns produced by the touch sensors, easing comparison.

Figure 3 depicts this metric graphically. Figure 3a shows that three robots with slightly different morphologies, using the same controller, produce sensor time series that rapidly diverge. However the sensor data for the two similar robots (indicated by the thick line and the line with circle markers) are similar for a longer initial time period (divergence around $t = 250$), compared to the divergence of the different robot from the two similar robots (around $t = 50$). The rolling mean metric then gives small values for morphologically similar robots, and higher values for morphologically different robots.

The genetic algorithm operating in the estimation phase is the same as that operating in the exploration phase, except for a few differences. This phase also evolves a population of 100 genomes, but the genomes are strings of 17 values (the simulator modifications) compared to strings of 68 values as in the exploration phase (the synaptic weights). Second, during subsequent passes through this phase, the initial population of random genomes is seeded with a copy of the genome with the best fitness (equation 1) obtained during the previous pass. Third, genome evaluation is different: the default simulation is modified according to the genome; the simulated robot is evaluated using the i previously evolved controllers; i sets of sensor time series are thus produced; and the genome's rolling mean is calculated. Selection, mutation and crossover are the same as in the exploration phase. The population of the estimation phase is also evolved for 30 generations during each pass, and outputs the best evolved simulator to the exploration phase, which begins again with

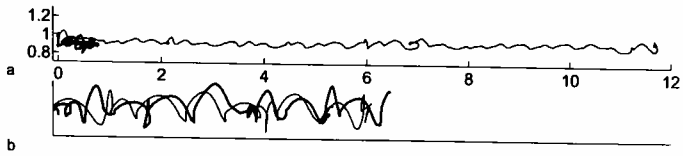

Figure 5: **Behavior recovery after controller transferal.** After the first pass through the exploration phase, the best evolved controller was used by the default simulated robot. The trajectory of its center of mass is given by the thin line in **a**. The same controller was then supplied to the target 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).

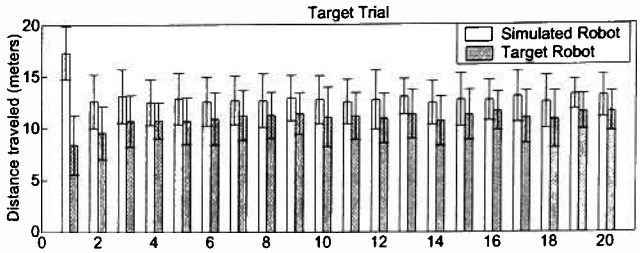

Figure 6: **Average transferal success after each target trial.** The light gray bars indicate the average distance traveled by the simulated robot using the best evolved controller output by that pass through the exploration phase, over all 50 runs. The dark gray bars indicate the average distance traveled by the target robot using the same controller, during each target trial. Error bars indicate two units of standard deviation.

a random population.

4) Termination Criteria: The algorithm iterates through the cycle shown in Figure 1a 20 times, starting at the exploration phase with the default simulator. This produces 20 evolved controllers (the fittest genome from each pass through the exploration phase), and 20 simulator modifications (the fittest genome from each pass through the estimation phase). The choice of 20 is arbitrary in this case; generally the algorithm continues until satisfactory behavior is attained on the target system, or until no simulator adaptations can explain the observed behavior. This latter failure can occur if either the evolutionary search cannot find a sufficiently accurate simulator, or if no combination of modifications can accurately describe the target system, implying that there is no point continuing in simulation.

Results

Fifty independent runs of the algorithm were conducted against the target robot outlined in Table 1. Figure 4 shows the 50 series of 20 best simulator modifications output after each pass through the estimation phase.

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 5a indicates the change in behaviors when the first evolved controller was transferred, and Figure 5b shows the behavior change when the 20th 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 6.

Discussion

Figure 4 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. This is most likely due to the relatively short time lags of the sensors near to those body parts, providing more information to the rolling mean metric (equation 1).

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 5a: the target robot simply moves randomly, and achieves no appreciable forward locomotion.

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 5b compared to Figure 5a). Also, the new gait causes the robot to hop (indicated by the large vertical curves of the robot's center of mass in 5b) instead of walk (indicated by the steady trajectory of Figure 5a). 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 6 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 6, 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 5b.

Conclusions

We have described a new algorithm—the estimation-exploration algorithm—for automatically improving a robot simulator using an evolutionary algorithm. The improvement is shown to be sufficient to allow for successful transferal of an evolved controller from simulation to a target robot, which exhibits some unknown differences compared to the default initial simulated robot.

This algorithm has several benefits: it is automatic; it achieves successful transferal after only a minimum of trials on the target robot; the sensors onboard the robot are used to generate behavior and provide indirect evidence about the morphological characteristics of the robot; and the algorithm is generalizable, as it has already been applied in other problem domains such as systems biology (Bongard and Lipson, 2004b) and robot damage diagnosis and recovery (Bongard and Lipson, 2004a).

Future experiments are planned in which the target robot, which is currently also simulated, will be replaced by an actual physical robot. Second, initial results (Bongard and Lipson, 2004a) have indicated that based on sensor feedback the virtual environment of the simulated robot can be automatically updated to reflect the target robot's physical environment. An interesting next step would be to equip the robot with rudimentary visual sensors, such as sonar, that could be used to automatically construct virtual analogues of obstacles or other agents in the robot's environment. We are currently formalizing methods for applying the algorithm to an arbitrary hidden nonlinear system. Source code, executables and documentation for verifying the experiments and extending the results reported here are available at `http://www.people.cornell.edu/pages/jb382/`.

Acknowledgements

This work was supported by the U.S. Department of Energy, grant DE-FG02-01ER45902. This research was conducted using cluster computing resources of the Cornell Theory Center.

References

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

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.

Bongard, J. C. and Lipson, H. (2004a). Automated robot function recovery after unanticipated failure or environmental change using a minimum of hardware trials. In *Proceedings of The 2004 NASA/DoD Conference on Evolvable Hardware*.

Bongard, J. C. and Lipson, H. (2004b). Automating genetic network inference with minimal physical experimentation using coevolution. In *Proceedings of The 2004 Genetic and Evolutionary Computation Conference*.

DiPaolo, E. A. (2000). Homeostatic adaptation to inversion of the visual field and other sensorimotor disruptions. In Meyer, J. A., Berthoz, A., Floreano, D., Roitblat, H. L., and Wilson, S. W., editors, *From Animals to Animats 6*, pages 440–449. MIT Press.

Floreano, D. and Mondada, F. (1998). Evolutionary neurocontrollers for autonomous mobile robots. *Neural Networks*, 11:1461–1478.

Floreano, D. and Urzelai, J. (2001). Neural morphogenesis, synaptic plasticity, and evolution. *Theory in Bioscience*, 120:225–240.

Funes, P. and Pollack, J. (1999). Computer evolution of buildable objects. In Bentley, P., editor, *Evolutionary Design by Computer*, pages 387–403. Morgan Kauffman, San Francisco.

Hornby, G. S. and Pollack, J. B. (2002). Creating high-level components with a generative representation for body-brain evolution. *Artificial Life*, 8(3):223–246.

Jakobi, N. (1997). Evolutionary robotics and the radical envelope of noise hypothesis. *Adaptive Behavior*, 6(1):131–174.

Lipson, H. and Pollack, J. B. (2000). Automatic design and manufacture of artificial lifeforms. *Nature*, 406:974–978.

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.

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.

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.

Sims, K. (1994). Evolving 3D morphology and behaviour by competition. *Artificial Life IV*, pages 28–39.

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.

Tokura, S., Ishiguro, A., Kawai, H., and Eggenberger, P. (2001). The effect of neuromodulations on the adaptability of evolved neurocontrollers. In Kelemen, J. and Sosik, P., editors, *Sixth European Conference on Artificial Life*, pages 292–295.