On the Origin of Environments by Means of Natural Selection

Moshe Sipper

■ The field of adaptive robotics involves simulations and real-world implementations of robots that adapt to their environments. In this article, I introduce *adaptive environmentics*—the flip side of adaptive robotics—in which the environment adapts to the robot. To illustrate the approach, I offer three simple experiments in which a genetic algorithm is used to shape an environment for a simulated KHEPERA robot. I then discuss at length the potential of adaptive environmentics, also delineating several possible avenues of future research.

The reasonable man adapts himself to the world; the unreasonable man persists to adapt the world to himself. Therefore, all progress depends on the unreasonable.

—George Bernard Shaw, 1903

An ant, viewed as a behaving system, is quite simple. The apparent complexity of its behavior over time is largely a reflection of the complexity of the environment in which it finds itself.

—Herbert A. Simon, 1969

The field of adaptive robotics studies the ways in which robots exhibiting some degree of autonomy adapt to their environments. Using both simulated and real robots, and applying techniques such as reinforcement learning, artificial neural networks, genetic algorithms, and fuzzy logic, researchers have obtained robots that display an amazing slew of behaviors and perform a multitude of tasks, including walking, pushing boxes, navigating, negotiating an obstacle course, playing ball, and foraging (Arkin 1998a).

To cite one typical example of an ever-growing many, Yung and Ye (1999) recently wrote:

We have presented a fuzzy navigator that performs well in complex and unknown environments, using a rule base that is learned from a simple corridor-like environment. The principle of the navigator is built on the fusion of the obstacle avoidance and goal seeking behaviors aided by an environment evaluator to tune the universe of discourse of the input sensor readings and enhance its adaptability. For this reason, the navigator has been able to learn extremely quickly in a simple environment, and then operate in an unknown environment, where exploration is not required at all.

This quote typifies the underlying theme of adaptive robotics: Have a robot adapt to a given environment. *Given* signifies neither that the environment is known nor that it is static; it means that the robot must adapt to the quirks and idiosyncrasies imposed by the environment—which, for its part, does nothing at all to accommodate the puffing robot.

This fundamental principle of adaptive robotics—the environment's unyielding nature—is repealed in this article. Dubbed *adaptive environmentics,* the basic idea is to create scenarios that are mirror images of those found in adaptive robotics: The environment adapts to a given robot.

I hasten to say that in some cases, it is not possible to alter the environment, and in other cases, having the robot adapt is simply the underlying objective. Adaptive robotics has produced many interesting results based on these principles. I believe, however, that considering the flip-side setup brings along its own bag of boons; this article aims to demonstrate qualitatively the benefits of adaptive environments and present possible avenues of explo-
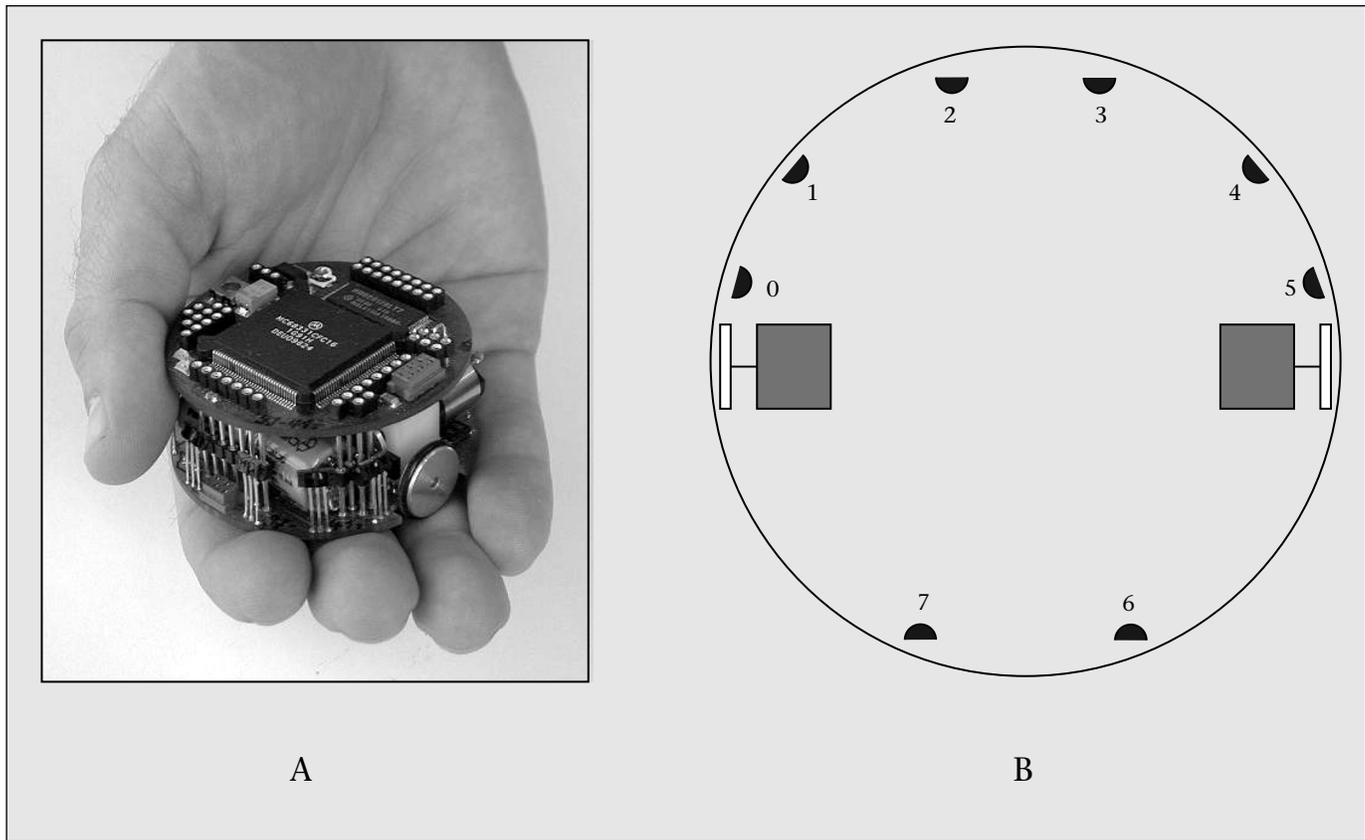
Figure 1. The Mobile KHEPERA Robot.

A. A diameter of 55 millimeters, a height of 30 millimeters, and a weight about 70 grams, the mobile KHEPERA robot is small enough to perform desktop experiments. B. It has 8 infrared and light sensors and two wheels, each controlled by a separate motor.

ration. (I further discuss the environment's mutability in Discussion and Future Work.)

The article is organized as follows: In the next section, I describe the experimental setup, consisting of a simulated, mobile KHEPERA robot whose environment is evolved using a genetic algorithm. The section entitled Results then describes the results of three experiments intended to illustrate the approach: (1) place lamps evolutionarily to guide a photophilic KHEPERA between two given points in an obstacle-ridden course; (2) lay out lamps as in the first experiment, such that both the tour time and the number of lamps are minimized; (3) optimize the lamp layout as in the second experiment, with the added possibility of repositioning obstacles in the form of walls. I conclude in Discussion and Future Work by discussing issues requiring further investigation and suggesting several possible avenues of future research.

Experimental Setup

My experiments involved a simulated version of the KHEPERA robot (Mondada, Franzi, and Ienne 1994),[1] a small, mobile device with eight surrounding infrared and light sensors and two separately controllable motors (figure 1). The simulator allows the user to write the control program for the robot and define its environment.[2]

In all the experiments described herein, I used a simple robot-control program, which essentially implements a photophilic Braitenberg vehicle (Braitenberg 1984). The program assigns speed values to both motors in accordance with the readings of the six frontal light sensors, resulting in a graded photophilic response: The robot is attracted to light, turning ever faster as it nears a light source (figure 2). The environment is a square of (simulated) size 1000 millimeters x 1000 millimeters (compared to the size of the real KHEPERA), in which one can place bricks and lamps (figure 3).

Because we are interested in adaptive environmentics, it is the environment (lamps and bricks) that will do the adapting, but the robot-control program remains fixed. It is with this goal in mind that I chose a simple Braitenberg vehicle, aiming to investigate how the environ-

ment will adapt to cope with it; the control program does not change at all. The adaptive algorithm used is a simple genetic algorithm (Vose 1999), the parameters of which are given in table 1 (I used the popular freeware software by Grefenstette.[3] For readers unfamiliar with genetic algorithms, consult the sidebar.)

Results

This section presents the results of the three experiments that illustrate my approach.

Experiment 1: Evolving Lamp Positions

In the first experiment, the robot is placed at the upper–left-hand corner of the brick environment (figure 3) and should find its way to the bottom-middle part. The evolutionary goal is to place seven lamps that act as way points, guiding the robot to its destination. The genome encodes 7 lamp positions, each defined by two 10-bit coordinates (the values [1000, 1023] are mapped to 999); thus, it is 140 bits long.

To assign a fitness score to an individual genome in the evolving population (that is, to a lamp layout), the seven lamps are first added to the environment. The photophilic robot is placed at position (100, 100), facing downward and rightward at an angle of 45 degrees (as in figure 3). It is then run for 2000 time steps (in one simulated time step, the robot advances about 1 to 2 millimeters or turns on its axis approximately 2 to 10 degrees—in accordance with the set motor-speed values). The fitness value, f, to be minimized is computed as

$$f = \min_{t=1}^{1000} \left\{ dist_t(650,500) \right\}$$

$$+ \min_{t=1001}^{2000} \left\{ dist_t(500,900) \right\}$$

where $dist_t(x, y)$ is the robot's Euclidean distance at time t from point (x, y). The fitness function thus measures how closely the robot passes two milestones—an intermediate one, (650,500), and the final one, (500, 900).[4]

I performed a total of 52 evolutionary runs,[5] the resulting average fitness of the best individual for each run being 23.1 (σ = 9.9). Of these, 42 runs produced an individual with fitness in the range 18 to 22 (that is, a total minimal distance to the two milestones of approximately 2 millimeters). Figure 4 shows the experimental results. Figure 4a shows two evolved lamp layouts and the trajectories taken by the robot.

```
left_motor =
    ( sensor[0]-550 + sensor[1]-550 + sensor[2]-550 +
      550-sensor[3] + 550-sensor[4] + 550-sensor[5] ) / 150;
if (left_motor>=0) left_motor=left_motor+NOMINAL_SPEED;
else               left_motor=left_motor-NOMINAL_SPEED;

right_motor =
    ( 550-sensor[0] + 550-sensor[1] + 550-sensor[2] +
      sensor[3]-550 + sensor[4]-550 + sensor[5]-550 ) / 150;
if (right_motor>=0) right_motor=right_motor+NOMINAL_SPEED;
else                right_motor=right_motor-NOMINAL_SPEED;
```

Figure 2. Control Program for the Photophilic Robot, Which Sets the Left and Right Motor Speed Values as a Function of the Six Frontal Light Sensors (Numbered 0–5).

Light readings are in the range 500 (dark) to 50 (near light source). Each motor can take on a speed value that is an integer in the range [–10, 10] . This control program produces a graded photophilic behavior, with the robot turning ever faster as it approaches a light source. The constant NOMINAL_SPEED, set to 2, induces forward movement when the light source is directly ahead or when no light is detected.

Parameter	Value
Number of Generations	200
Population Size	100
Crossover Rate	0.9
Mutation Rate	0.01
Selection Method	Rank based
Elitism	Yes (best individual survives intact to next generation)
Genome Size (bits)	140 (experiment 1) 200 (experiment 2) 200 (experiment 3)
Fitness Function	Experiment dependent (see text)

Table 1. Parameters of the Genetic Algorithm.

Experiment 2: Minimizing Lamp Count and Tour Time

The lamp-layout task studied in the previous subsection is, in fact, simple enough to carry out manually by tinkering with lamp positions using the simulator. The intent in this first experiment was to provide a preliminary example of adaptive environmentics. Moreover, the lamp-layout task proved a good benchmark problem for gaining insight into the adaptive scenario, as do the simple XOR computation for artificial neural networks (Rumelhart, Hinton, and Williams 1986) and the number-of-ones fitness function for genetic algorithms (Michalewicz 1996). In this section, we study a veritable hard task involving multicriterion optimization: laying

Genetic Algorithms

The idea of applying the biological principle of natural evolution to artificial systems, introduced more than four decades ago, has seen impressive growth in the past few years. Usually grouped under the terms *evolutionary algorithms* or *evolutionary computation,* we find the domains of genetic algorithms, evolution strategies, evolutionary programming, and genetic programming (Fogel 1995; Koza 1992; Michalewicz 1996). Such algorithms are common nowadays, having been successfully applied to numerous problems from different domains, including optimization, automatic programming, machine learning, economics, medicine, ecology, population genetics, and hardware design. In this article, I consider the evolutionary methodology known as *genetic algorithms.*

A genetic algorithm is an iterative procedure that involves a population of individuals, each one represented by a finite string of symbols, known as the genome, encoding a possible solution in a given problem space. This space, referred to as the *search space,* comprises all possible solutions to the problem at hand. Genetic algorithms are usually applied to spaces that are too large to be exhaustively searched.

The standard genetic algorithm proceeds as follows: An initial population of individuals is generated at random or heuristically. Every evolutionary step, known as a *generation*, the individuals in the current population are decoded and evaluated according to some predefined quality criterion, referred to as the *fitness,* or *fitness function.* To form a new population (the next generation), individuals are selected according to their fitness. Many

```
begin GA
    g := 0 {generation counter}
    Initialize population P(g)
    Evaluate population P(g)         (that is, compute fitness values)
    while not done do
            g := g + 1
            Select P(g) from P(g – 1)
            Crossover P(g)
            Mutate P(g)
            Evaluate P(g)
    end while
end GA
```

Figure A. Pseudocode of the Standard Genetic Algorithm

selection procedures are currently in use, one of the simplest being *fitness-proportionate selection*, where individuals are selected with a probability proportional to their relative fitness. This approach ensures that the expected number of times an individual is chosen is approximately proportional to its relative performance in the population. Another form of selection is rank based, wherein the individuals in the population are ranked according to fitness, and the expected value of each individual depends on its rank rather than its absolute fitness (this selection method is used to prevent premature convergence). In both cases, high-fitness ("good") individuals stand a better chance of "reproducing," but low-fitness ones are more likely to disappear.

Selection alone cannot introduce any new individuals into the population; that is, it cannot find new points in the search space. New individuals are generated by genetically inspired operators, of which the most well known are crossover and mutation. Crossover is performed with probability p_c

(the *crossover probability* or *crossover rate*) between two selected individuals, called *parents*, by exchanging parts of their genomes (that is, encodings) to form two new individuals, called *offspring.* In its simplest form, substrings are exchanged after a randomly selected crossover point. This operator tends to enable the evolutionary process to move toward promising regions of the search space. The mutation operator is introduced to prevent premature convergence to local optima by randomly sampling new points in the search space. It is carried out by flipping bits at random, with some (usually small) probability p_m. Genetic algorithms are stochastic iterative processes that are not guaranteed to converge. The termination condition can be specified as some fixed, maximal number of generations or as the attainment of an acceptable fitness level. Figure A presents the standard genetic algorithm in pseudocode format.

out lamps in the environment of figure 3 to guide the robot from point (100, 100) to point (500, 900), such that both the tour time and the number of lamps are minimized.

The genome is similar in structure to that of the previous subsection: It encodes a maximum of 10 lamp positions, each defined by two 10-bit coordinates and, thus, is 200 bits long. A value of either the *x* or *y* coordinate in the range [1000, 1023] is taken to mean that the lamp in question is not used (this representation is more mutation immune and, thus, less brittle than simply allotting one bit to encode a lamp's functional-nonfunctional status; a more sophisticated approach would be to use dynamically modifiable genomes, such as variable-length representations (Wu and Banzhaf 1998) and tree encodings (Banzhaf et al. 1997).

The fitness value, *f*, is computed as

$$f = \min_{t=1}^{1000}\left\{dist_t(650,500)\right\} + \alpha t^1_{min}$$
$$+ \beta \min_{t=1001}^{2000}\left\{dist_t(500,900)\right\} + \alpha\left(t^2_{min} - 1000\right)$$
$$+ \gamma\, N_{lamps}$$

where $dist_t(x, y)$ is the robot's Euclidean distance at time t from point (x, y); t^1_{min} and t^2_{min} are, respectively, the time steps at which the minimal distances from the first and second milestones are attained; N_{lamps} is the number of lamps used; and α, β, and γ are constants set empirically (through trial and error) to α = 0.2, β = 1.5, γ = 100.

In general, because there are three criteria to optimize—(1) time to milestones, (2) distance to milestones, and (3) number of lamps—which are in conflict because of the nature of the obstacle-laden course, there is no single optimal solution—only an ensemble of Pareto-optimal ones. Such a multicriterion-optimization problem might benefit from recent advances in evolutionary algorithms (Deb and Horn 2000), thus obviating the need for setting α, β, and γ as such.

I performed a total of 54 evolutionary runs, observing that the genetic algorithm converged mainly toward three types of solution: (1) 1 to 3 lamps guiding the robot straight down until it hits the wall situated along the straight-trajectory (and fastest) path to the final goal, (2) 4 to 5 lamps guiding the robot to one of the walls, and (3) 4 to 5 lamps guiding the robot to the goal (Figure 4b shows two solutions of this latter type). The resulting average fitness of the best individual for a run was 806.06 (σ = 76.03).

Experiment 3: Interior Design and Redesign

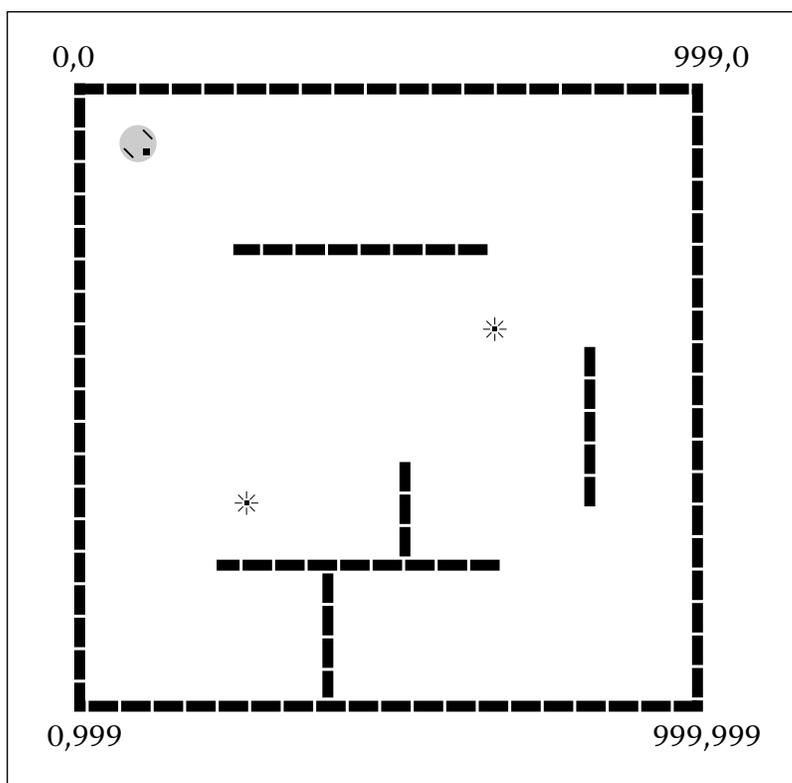In the final experiment, the goal is similar to that of the previous subsection: Lay out lamps

Figure 3. Sample Environment with Brick Walls and Two Lamps.
The KHEPERA is positioned at coordinates (x, y) = (100, 100), facing downward and rightward at an angle of 45 degrees.

in the environment of figure 3 to guide the robot from point (100,100) to point (500,900), such that both the tour time and the number of lamps are minimized; this time, however, the genetic algorithm is "allowed" not only to design (lamp layout) but also to redesign: The five inner walls can be repositioned. The genome is 200 bits long: The first 100 bits encode as many as 5 lamp positions as in the previous subsection (the maximum lamp count was lowered from 10 to 5 because of the results of the preceding experiment). The last 100 bits encode the upper–left-hand coordinates of the five walls: 2 horizontal walls—comprising, respectively, 8 and 9 bricks—and 3 vertical walls—comprising, respectively, 3, 4, and 5 bricks (decoded coordinates representing a wall that extends beyond the enclosure borders are reset to the maximal values still within range). The fitness function is identical to that of the previous subsection.

Having performed 50 evolutionary runs, I observed that the genetic algorithm took advantage of the interior-redecoration option: The resulting average fitness of the best individual for a run was 351.04 (σ = 3.33), as opposed to 806.06 for experiment 2, with all layouts containing a single lamp. Figure 4c
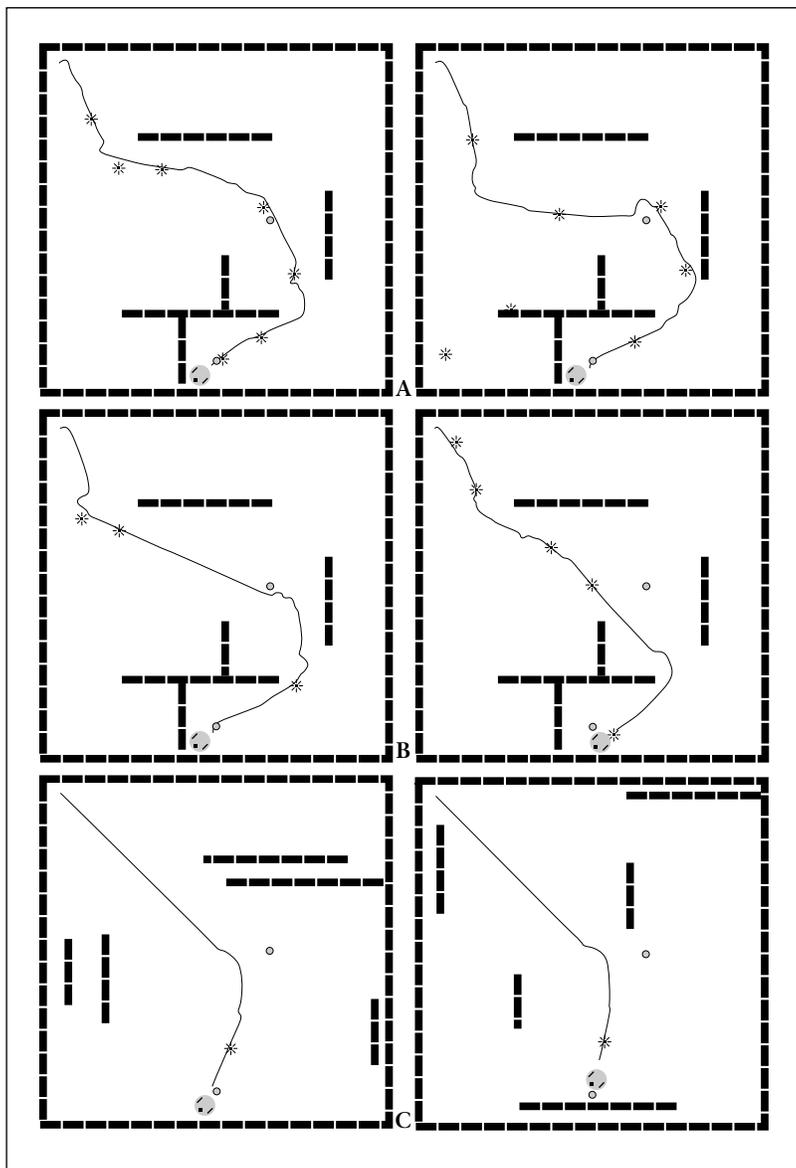
*Figure 4. Experimental Results of Evolved Lamp Layouts
and the Robot Trajectories They Occasion.*

A. Experiment 1, where the evolutionary goal is to place seven lamps to guide the photophilic robot from the upper–left-hand point (100, 100) to the bottom-middle point (500, 900). The two milestones—(650, 500) and (500, 900)—are also shown. The left layout has fitness f = 18.27, and the right layout has fitness f = 18.98. B. Experiment 2, where the evolutionary goal is to lay out lamps as in the first experiment, such that both the tour time and the number of lamps are minimized. An evolved four-lamp solution is shown to the left, and an evolved five-lamp solution is shown to the right. Compared with A, the trajectories are shorter, and the layouts contain less lamps. C. Experiment 3, which is identical to the second experiment except that the genetic algorithm is "allowed" to reposition the five inner walls. Both layouts contain a single lamp.

shows two evolved lamp layouts and the trajectories taken by the robot, clearly demonstrating how the evolutionary architect has simply chucked the walls out of the robot's way. More complex layout specifications can readily be envisioned, such as requiring that the walls form three substructures as in the original envi-ronment (figure 3). Such scenarios provide an interesting twist on those where the robot avoids the obstacles: In this case, the obstacles avoid the robot.

Discussion and Future Work

The experiments presented in the previous section are but a first foray into adaptive environmentics. There are many issues that require further investigation as well as several open avenues of future research that suggest themselves. These I discuss in this section.

Robustness

One of the major problems with adaptive techniques, especially when applied to noise-abundant domains such as robotics, is the robustness of the obtained solutions (Ronald and Sipper 2000). For example, a robot placed in an evolved (or hand-designed) environment, such as those shown in figure 4, can at times wander off the (expected) beaten path. In some cases, a "robustness patch" can be had quite simply by incorporating the robustness criteria into the cost function. This is, in fact, an advantage with techniques such as evolutionary algorithms: One can augment the cost function to explicitly embody harsher specifications. In these experiments, for example, the robot could be run several times to evaluate a single individual in the evolving population, rendering evolved layouts more noise immune (to reduce computational cost, I only performed one trial run for each evaluation).[6]

Control Programs

Although I used a simple photophilic robot-control program, more sophisticated "brains" can be imagined. One could fuse several behaviors, based on different sensors (infrared, ultrasonic, visual, tactile), in a subsumptionlike manner (Brooks 1986). Artificial neural networks, fuzzy systems, case-based reasoning, and other AI techniques are also prime candidates for enhancing the robot's capacities (Arkin 1998a). It would be interesting to test how environments adapted to these more complex control schemes. (I note in passing that I also experimented with a photophobic robot, where lamps act as repellents rather than attractors. Much like a cart, which is easier to pull using a pole than to push, the photophobic robot proved harder to work with, inducing me to introduce the photophilic version.)

Functional-Envelope Evaluation

Given a robot controller, one often needs to derive its functional envelope, namely, the

environmental parameters' ranges within which the robot functions correctly. Questions might include the range of lighting conditions the robot's sensors can handle, the maximal angle of a corner such that the robot still detects it as such, and infrared-reflection limitations. When analytic methods prove insufficient, adaptive environmentics might aid in the evaluation of the robot's functional envelope.

Collective Robotics

The scenarios studied herein comprised an environment adapting to a single robot; a natural extension would be to study multirobot systems (Arkin 1998b). For example, given a large surface—such as a hospital interior or a supermarket floor—to be cleaned by an army of robocleaners, one might evolve navigational landmarks, evolve the positions of trash bins to minimize the robots' overall time spent in garbage-disposal trips, and perhaps further optimize the layout of electricity outlets.

Adaptive Envirobotics

With adaptive environmentics, as presented herein, the environment is the one doing the adapting, and the robot's behavioral program is immutable. This latter restriction might be lifted, creating a system in which both the environment and the robot coevolve (Paredis 1995; Peña-Reyes and Sipper 2000; Potter and de Jong 2000). This approach might engender better solutions in less time; for example, with the ability to coadapt to its evolving environment (for example, by way of evolution or learning), our robot might ultimately be able to negotiate the terrain more easily. (I note in passing that on a much grander scale—according to the Gaia hypothesis—earth's atmospheric conditions not only bring about terrestrial evolution but indeed coevolve with the planet's fauna and flora [Lovelock 1995]). Moreover, the robot might modify the environment to suit itself, as with a person moving and arranging furniture in a newly rented apartment or a beaver damming a stream (Hammond, Converse, and Grass 1995; Kirsh 1996).

Simulated Versus Real

There has been a long ongoing debate in robotics on the issue of simulation versus the real world. Although evolving a robot's control program can often be done in a real-world experiment, adaptive environmentics presents a harder case—though not an impossible one. For example, the previous experiments could be carried out with a real KHEPERA, a grid floor with regularly placed lights that can be turned on and off, and mechanically configurable

bricks. On the whole, though, it seems that (at least for now) simulated adaptive environmentics is easier, as with recent works on the evolution of physical structures (Funes and Pollack 1998) and robot morphologies (Pollack et al. 2000): Evolving every individual in the real world is too complicated and too costly, although the best solutions might well be transferred from simulation to the real world (as done by Pollack et al. [2000]). (It is worth mentioning in the context of simulation versus reality Jakobi's recent work on minimal simulations, where the goal is to design simulations that are easier to construct, run faster, and transfer readily to reality [Jakobi 1998].)

Using other Machine Learning Techniques

Although I have concentrated herein on adapting environments using evolutionary algorithms, other machine-learning methods might be brought to bear on the problem. Incremental approaches that are not population based—for example, artificial neural networks and reinforcement learning—might represent the advantage of easier amenability to implementation in the real world. For example, a neural network might be used to tune the signal of a sonar-guiding beacon or the light level of a lamp post to adapt to the quirks of a robot's sensors.

Applications

Adaptive robotics finds applications in areas where the robot is placed in a given environment and must exhibit robust behavior. The applications of adaptive environmentics will be found in those areas in which it is possible and propitious to forge the environment, possibly using standard, commercial, off-the-shelf devices. Example applications that come to mind are (1) placing disposal bins for a cleaning robot such that time to bin is minimized, (2) designing locations of repair-refueling stations for planetary explorers, (3) laying out guidance beacons (for example, sonar) to aid navigation of underwater probes, and (4) designing sewer systems to facilitate the operation of sewer robots (Kirchner and Hertzberg 1997).

The ability to mold a robot's environment using adaptive techniques complements the current effort in adaptive robotics and will hopefully help advance the state of the art of the field. As noted by Simon (1981, p. 75), "goal-directed behavior simply reflects the shape of the environment in which it takes place…." It is sobering to reflect on the behavioral system Simon (1981, p. 75) is addressing: …in large part human goal-directed behavior

simply reflects the shape of the environment in which it takes place....

Acknowledgments

Notes

1. The K-Team home page, www.k-team.com.

2. O. Michel, KHEPERA simulator package version 2.0, 1996, diwww.epfl.ch/lami/team/ michel/khep-sim/SIM2.tar.gz.

3. J. J. Grefenstette, GENESIS (genetic search implementation system) version 5.0, 1994, www.aic.nrl.navy.mil/pub/galist/src/genesis.tar.Z.

4. Preliminary experiments revealed that an intermediate milestone was needed to boost the evolutionary process.

5. Running on a Sun ULTRA10 at 333 megahertz, a single fitness evaluation took approximately 0.22 seconds, and an entire run (200 generations, 100 individuals) took about 75 minutes.

6. Note that the simulator used incorporates random noise, and thus, the robot's behavior, even when started in the same configuration, is not deterministic (see note 2).

References

Arkin, R. C. 1998a. Adaptive Behavior. In *Behavior-Based Robotics,* 305–357. Cambridge, Mass.: MIT Press.

Arkin, R. C. 1998b. Social Behavior. In *Behavior-Based Robotics,* 359–420. Cambridge, Mass.: MIT Press.

Banzhaf, W.; Nordin, P.; Keller, R. E.; and Francone, F. D. 1997. *Genetic Programming—An Introduction: On the Automatic Evolution of Computer Programs and Its Applications.* San Francisco, Calif.: Morgan Kaufmann.

Braitenberg, V. 1984. *Vehicles: Experiments in Synthetic Psychology.* Cambridge, Mass.: MIT Press.

Brooks, R. A. 1986, A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation* RA-2(1): 14–23.

Deb, K., and Horn, J., eds. 2000. *Evolutionary Computation* (Special Issue on Multicriterion Optimization) 8(2).

Fogel, D. B. 1995. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* New York: IEEE Computer Society Press.

Funes, P., and Pollack, J. 1998. Evolutionary Body Building: Adaptive Physical Designs for Robots. *Artificial Life* 4(4): 337–357.

Hammond, K. J.; Converse, T. M.; and Grass, J. W. 1995. The Stabilization of Environments. *Artificial Intelligence* 72(1–2): 305–327.

Jakobi, N. 1998. Minimal Simulations for Evolutionary Robotics. Ph.D. dissertation, School of Cognitive and Computing Sciences, University of Sussex at Brighton.

Kirchner, F., and Hertzberg, J. 1997. A Prototype Study of an Autonomous Robot Platform for Sewerage System Maintenance. *Autonomous Robots* 4(4): 319–331.

Kirsh, D. 1996. Adapting the Environment Instead of Oneself. *Adaptive Behavior* 4(3–4): 415–452.

Koza, J. R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, Mass.: The MIT Press.

Lovelock, J. 1995. *The Ages of GAIA: A Biography of Our Living Earth.* 2d ed. Oxford, U.K.: Oxford University Press.

Michalewicz, Z. 1996. *Genetic Algorithms + Data Structures = Evolution Programs.* 3d ed. Heidelberg, Germany: Springer-Verlag.

Mondada, F.; Franzi, E.; and Ienne, P. 1994. Mobile Robot Miniaturization: A Tool for Investigation in Control Algorithms. In *Experimental Robotics III: The Third International Symposium,* eds. T. Yoshikawa and F. Miyazaki, 501–513. Lecture Notes in Control and Information Sciences Volume 200. London: Springer-Verlag.

Paredis, J. 1995. Coevolutionary Computation. *Artificial Life* 2(4): 355–375.

Peña-Reyes, C.-A., and Sipper, M. 2000. Applying Fuzzy CoCo to Breast Cancer Diagnosis. In Proceedings of the 2000 Congress on Evolutionary Computation (CEC00), Volume 2, 1168–1175. New York: IEEE Computer Society.

Pollack, J. B.; Lipson, H.; Ficici, S.; Funes, P.; Hornby, G.; and Watson, R. 2000. Evolutionary Techniques in Physical Robotics. In *Evolvable Systems: From Biology to Hardware, Proceedings of the Third International Conference (ICES2000),* eds. J. Miller, A. Thompson, P. Thomson, and T. C. Fogarty, 175–186. Lecture Notes in Computer Science Volume 1801. Heidelberg, Germany: Springer-Verlag.

Potter, M. A., and De Jong, K. A. 2000. Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents. *Evolutionary Computation* 8(1): 1–29.

Ronald, E. M. A., and Sipper, M. 2000. Engineering, Emergent Engineering, and Artificial Life: Unsurprise, Unsurprising Surprise, and Surprising Surprise. In *Artificial Life VII: Proceedings of the Seventh International Conference on Artificial Life,* eds. M. A. Bedau, J. S. McCaskill, N. H. Packard, and S. Rasmussen, 523–528. Cambridge, Mass.: MIT Press.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1986. Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing, Volume 1: Foundations,* eds. D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, 318–362. Cambridge, Mass.: MIT Press.

Simon, H. A. 1981. *The Sciences of the Artificial.* 2d ed. Cambridge, Mass.: The MIT Press.

Vose, M. D. 1999. *The Simple Genetic Algorithm: Foundations and Theory.* Cambridge, Mass.: MIT Press.

Wu, A. S., and Banzhaf, W., eds. 1998. *Evolutionary Computation* (Special Issue on Variable-Length Representation and Noncoding Segments for Evolutionary Algorithms) 6(4).

Yung, N. H. C., and Ye, C. 1999. An Intelligent Mobile Vehicle Navigator Based on Fuzzy Logic and Reinforcement Learning. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics* 29(2): 314–321.

Moshe Sipper is a senior lecturer in the Department of Computer Science at Ben-Gurion University, Israel, and a visiting senior researcher in the Logic Systems Laboratory at the Swiss Federal Institute of Technology in Lausanne. His chief interests involve the application of biological principles to artificial systems, including evolutionary computation, cellular computing, bio-inspired systems, evolvable hardware, complex adaptive systems, artificial life, fuzzy logic, and artificial neural networks. Sipper has published over 90 research publications in these areas as well as the book *Evolution of Parallel Cellular Machines: The Cellular Programming Approach* (Springer-Verlag, 1997). His e-mail address is sipper@cs.bgu.ac.il.