Evolutionary Robotics and the Radical Envelope-of-Noise Hypothesis

Nick Jakobi

Adaptive Behavior 1997 6: 325
DOI: 10.1177/105971239700600205

Evolutionary Robotics and the Radical Envelope-of-Noise Hypothesis

Nick Jakobi*
University of Sussex

For several years now, various researchers have endeavored to apply artificial evolution to the automatic design of control systems for real robots. One of the major challenges they face concerns the question of how to assess the fitness of evolving controllers when each evolutionary run typically involves hundreds of thousands of such assessments. This article outlines new ways of thinking about and building simulations upon which such assessments can be performed. It puts forward sufficient conditions for the successful transfer of evolved controllers from simulation to reality and develops a potential methodology for building simulations in which evolving controllers are forced to satisfy these conditions if they are to be reliably fit. It is hypothesized that as long as simulations are built according to this methodology, it does not matter how inaccurate or incomplete they are: Controllers that have evolved to be reliably fit in simulation still will transfer into reality. Two sets of experiments are reported, both of which involve minimal look-up table–based simulations built according to these guidelines. In the first set, controllers were evolved that allowed a Khepera robot to perform a simple memory task in the real world. In the second set, controllers were evolved for the Sussex University gantry robot that were able to distinguish visually a triangle from a square, under extremely noisy real-world conditions, and to steer the robot toward the triangle. In both cases, controllers that were reliably fit in simulation displayed extremely robust behavior when downloaded into reality.

Key Words: simulations; evolutionary robotics; robot-environment interactions; neural networks

Introduction

The artificial evolution of control architectures typically involves the constant and repetitive testing of hundreds upon thousands of individuals as to their ability to behave in a certain way or to perform a certain task. In the case of real robots, this testing procedure is far from a trivial matter and [with the exception of certain hybrid approaches (Thompson, 1995; Nolfi, Floreano, Miglino, & Mondada, 1994a)] can

* School of Cognitive and Computing Sciences, University of Sussex, Brighton BN1 9QH, England; E-mail: nickja@cogs.susx.ac.uk

be done in only one of two ways: Control architectures either must be evaluated on real robots in the real world, or they must be evaluated in simulations of real robots in the real world. Both of these approaches are problematic.

The evaluation of control architectures for real robots must be done in real time, which makes the entire evolutionary process prohibitively slow (Matarić and Cliff, 1996). For example, Matarić and Cliff (1996) cite the evolution of collision-free navigation in a Khepera robot, which, in the experiments reported by Floreano and Mondada (1994), took a total of 65 hours (100 generations at 39 minutes per generation) to evolve.[1] It is difficult to see how this approach can scale up, if the behaviors in which we are interested require thousands or even millions of generations. Nonetheless, even if we are resigned to an evolutionary process that takes years rather than days, then there are other problems that must be faced. The process must be automated, which begs questions about such concerns as how data are to be collected for fitness evaluations and how the robot will be placed at the start of fitness trials without human intervention. Power must be supplied continuously to robots in situations in which batteries have limited life spans and tethering by a permanent power lead is not always possible. In addition, machines break down, especially under the sort of continuous random battering that the real-world evaluation approach advocates. Clearly, the alternative simulation approach would be preferable, as it avoids all these problems and can run at faster than real time.

Several experimenters, including Jakobi, Husbands, and Harvey (1995), Beer and Gallagher (1992), and Miglino, Lund, and Nolfi (1995) have shown that it is possible to evolve in simulation control architectures for a real robot. Now that this is no longer in doubt, the question becomes one of whether the technique will scale up. In Matarić and Cliff (1996), the authors argue that if behavioral transference can be guaranteed only when a carefully constructed, empirically validated simulation is used, then as robots and the behaviors we want to evolve for them become more complicated, so do the simulations. The level of complexity involved, they argue, would make such simulations so computationally expensive that all speed advantages over real-world evolution are lost and so difficult to design that the time taken in development outweighs time saved by fast evolution.

Clearly, the main challenge for the simulation approach to evolutionary robotics is to invent a general theoretical basis and methodology on which fast-running simulators can be easily and cheaply built that guarantee the transference of evolved behaviors from simulation to reality.

This article puts forward such a theoretical and methodological basis, albeit at

1 The shape-discrimination behavior evolved in Harvey, Husbands, and Cliff (1994) took only 36 hours to evolve, but this is of the same order of magnitude.

a preliminary stage, and outlines some notable experimental successes using the proposed techniques. Section 2 undertakes a conceptual analysis of how it is possible for control architectures that have evolved in simulation to transfer into reality. Two conditions on evolved controllers are posited that together are sufficient to ensure that this transference is successful. Section 3 outlines a methodology for building simulations that force reliably fit control architectures to meet these two conditions and therefore to cross the reality gap successfully. The concept of fast, easy-to-build, minimal simulations also is introduced in this section. Section 4 outlines evolutionary experiments that involve a minimal simulation of a Khepera robot, and section 5 outlines evolutionary experiments that involve a minimal simulation of the Sussex University gantry robot. Finally, section 6 offers some conclusions and thoughts for the future.

How Is It Possible to Cross the Reality Gap?

As has been demonstrated in several reports (Jakobi et al., 1995; Beer & Gallagher, 1992; Nolfi, Miglino, & Parisi, 1994b), it *is* possible to evolve control architectures in simulation for a real robot. However, the explanations offered by the authors of these articles as to why behaviors successfully transfer to reality when evolved under certain simulation conditions though not under others fall well short of the level of understanding necessary to develop a general simulation-building methodology. The consensus view seems to be that control architectures will transfer successfully if the right amount of noise is included in a carefully constructed and empirically validated simulation of the robot and its environment.[2] However, there is no such thing as the perfect simulation; some real-world features will be modeled at the expense of others, and because *our* empirically validated simulation might be *your* unrealistic toy-world, we cannot agree on what to put into the simulation and what to leave out of it without objective criteria based on a sound theoretical understanding.

2.1 What constitutes success?

If we are to devise a general methodology for building simulations for evolutionary robotics, it is important to define exactly what we mean when we say that a behavior has successfully transferred from simulation to reality. In Miglino et al. (1995), the authors look at the fitnesses of control architectures in simulation and compare them to the fitnesses of the control architectures in reality but, as we shall see in section 3.4, this is not always possible when using the type of simulations proposed in this article,

2 Although the nature of the "right amount of noise," and indeed even what it means for a behavior to "transfer successfully from simulation to reality," varies markedly among articles on the topic.

and so we shall not be using this criterion here. In Jakobi et al. (1995), the authors use a more subjective approach to judge whether control architectures behave in reality in a manner that is qualitatively similar to how they behave in simulation, but again, as we shall see, this is not always possible.

For the purposes of this article, a control architecture is said to have successfully crossed the reality gap if it successfully and reliably displays the behavior it was evolved to display when downloaded into reality. It should be stressed that this does not involve any comparison between the behavior of the control architecture in simulation and in reality, but merely a judgment, based on thorough testing, of whether the control architecture reliably performs the task in reality. As in Jakobi et al. (1995), this often may be a somewhat subjective measure but should nevertheless be unambiguous. If one uses a simulation to evolve robot controllers to move around a cluttered environment while avoiding objects, for example, then control architectures successfully cross the reality gap if, when downloaded, they do indeed cause a robot, in a reliable fashion, to move around a cluttered environment while avoiding objects. If one uses a simulation to evolve visually guided robot controllers that steer toward a target, then control architectures successfully cross the reality gap if, when downloaded, they do indeed reliably steer the robot toward the target.

2.2 Overcoming the failings of simulation

All the worries and problems associated with getting controllers to cross the reality gap spring from one simple fact: It is not possible to build a simulation that is a perfect copy of the real world. If it were, then from the point of view of evolving robot controllers, there would be no differences between simulation and reality, and we would be surprised if they *did not* cross the reality gap. Unfortunately, any real-life simulation will differ from a perfect copy of the real world on two counts: It will model only a finite set of real-world features and processes, and those features and processes that it does model, it will model inaccurately. Both of these failings have fundamental implications for the ways that we should think about simulations for evolutionary robotics and the conditions that must be satisfied if evolved robot controllers are to cross the reality gap. We will look at both failings in turn.

2.2.1 Simulations cannot accurately model everything Because a simulation can model only a finite set of real-world features and processes, the first thing that we must decide when building a simulation is exactly what feature and processes should constitute this set. Obviously, for the purposes of evolutionary robotics, we are interested only in those real-world features and processes that have some bearing on a particular robot or robots and a particular environment. It still remains to be determined, however, which of these choices we must model and which we can get

away with ignoring, in order that evolved controllers will cross the reality gap. There is little point in modeling the color of a robot or the objects in its environment, for example, if that robot is fitted only with sonar range sensors.

The approach advocated in this article is very different from the conventional one. It is based on the observation that the only aspects of a simulation that have to be modeled accurately, if we are to ensure that a particular robot control architecture will cross the reality gap, are those that the particular robot control architecture uses and relies on to perform its behavior. In other words, if we accurately model only a subset of all the possible robot-environment interactions, then provided that the behaviors of evolving robot controllers can be constrained to depend on the members of this subset and this subset alone, from their point of view there will again be no difference between simulation and reality. If this subset is sufficient to underlie a particular behavior that we are interested in evolving, then robot controllers that evolve in this way to perform the task in question will transfer successfully from simulation into reality.

The conventional approach is to try to model, as accurately as possible, *all* the real-world features and processes that could conceivably affect a robot's behavior. The rationale behind this is that, again (ignoring the inevitable inaccuracy of the modeling process for the time being), there will be no differences between simulation and reality from the point of view of evolving robot controllers; we can therefore give evolution a free rein, safe in our knowledge that whatever aspects of the simulation on which evolving controllers come to depend, those aspects also will be present in the real world. In practice, however, no matter how comprehensive the model, there will always be real-world features that have been left out. Even with extensive and time-consuming empirical validation, simulations built according to this approach can hope to capture only a subset of the totality of possible robot-environment interactions. They should therefore be thought of in the same way as the simulations discussed earlier.

Because these concepts are central to the ideas presented later in this article, we will give them a special terminology.[3] Throughout the rest of the article, the set of robot-environment interactions that we decide to model as a basis for the behaviors of evolving controllers will be referred to as the *base set* of robot-environment interactions. Such a base set might include the way in which infrared sensors interact with nearby objects to return values, for example, or the way in which wheel speeds respond to motor signals to move the robot around the environment, or the way in which a camera returns an image when pointed at a particular feature in the environ-

3 A formal and theoretical framework that offers a slightly different viewpoint is developed in Jakobi (1997a). This formal framework, which is more amenable to mathematical analysis than to an intuitive understanding, should be seen as complimenting the terminological framework offered here.

ment. Those aspects of the simulation that correspond to members of this base set, and on which evolving controllers can safely depend and that they can use without encountering any differences between simulation and reality, will be referred to as the *base set aspects* of the simulation. Also, there likely will be aspects of the simulation on which behaviors of evolving controllers can depend but that have no basis in reality. These will derive from the simple fact that a simulation must be a coherent whole from the point of view of evolving control architectures. In other words, if only a few robot-environment interactions are modeled accurately in the simulation, then those that are not modeled often will leave "gaps" that must be filled in by arbitrary values and processes so that the simulation constitutes a complete virtual reality. Those aspects of the simulation that are not base set aspects, but on which evolving control architectures can come to depend nevertheless, will be referred to throughout the rest of the article as the *implementation aspects* of the simulation. We will examine concrete examples of all of these terms in section 3.

To see why the selection of certain concepts and the assignment of special terms is important, consider a *hypothetical* simulation in which we have decided on a particular base set of robot-environment interactions and have modeled them 100 percent accurately. In other words, we have created a simulation the base set aspects of which are 100 percent accurate. What are the conditions under which controllers evolved in such a simulation will transfer to reality? The answer is that if a control architecture evolves to depend on the base set aspects of the simulation to perform its behavior, *and those base set aspects alone* (either implicitly or explicitly), then the control architecture will successfully cross the reality gap, the simple reason being that, from the point of view of such a control architecture, there will be no difference between this simulation and one that is a perfect copy of the entire real-world situation.

Control architectures whose behaviors are exclusively grounded in the base set aspects of a particular simulation will be referred to throughout the article as being *base set exclusive*. Those controllers that are not base set exclusive but that depend on implementation aspects of the simulation and that have no counterpart in reality will, more likely than not, fail when downloaded. Their behaviors will depend on things that may or may not be true of the real world.

2.2.2 Simulations cannot accurately model anything Even if we know the conditions under which robot controllers will cross from a simulation whose base set aspects are 100 percent accurate into reality, we are only half-way toward a general explanation. It just is not possible to model a base set of robot-environment interactions with 100 percent accuracy. To explain how control architectures can and do

cross the reality gap, we need to add something to the exclusivity condition laid out earlier. We must state the conditions under which a control architecture will transfer into reality from a simulation whose base set aspects are *inaccurate*.

Consider first that, as defined previously, for a control architecture to cross the reality gap it need not perform the task identically in the real world to the way that it performs the task in simulation, but it must perform the task in the real world. Thus, small inaccuracies in the base set features of a simulation might result in slight differences between a controller's behavior in simulation and its behavior in reality, but as long as it continues to behave *satisfactorily* in reality, then we may say that the control architecture has successfully crossed the reality gap. This is akin to hitting a barn door with a shotgun at five paces: If your aim is off by a meter or so, you will still hit the barn door, and this is all we are after. Of course, in more complicated and involved situations, there will not be so much room to maneuver, but it should be kept in mind that even for the most delicate of behaviors, there will normally be a little bit of leeway in which small discrepancies are lost.

Some control architectures, however, will be more robust to inaccuracy than are others. The level of base set inaccuracy that an evolved control architecture can tolerate before it ceases to perform satisfactorily in the real world will depend on exactly *how* it uses the base set aspects of the simulation to perform the task. For example, it has long been appreciated in the engineering world that processes that employ feedback or similar techniques will be far more robust to inaccuracy and noise than those that do not (Brogan, 1991). This is true here also: Nonbrittle control strategies and behaviors that constantly correct themselves as they go, either through explicit feedback loops or implicitly via the environment [as in Braitenberg's vehicles (Braitenberg, 1984)], lend themselves to the handling of the differences between simulation and reality. However, in certain situations, even the most brittle, ballistic of control strategies will perform the task satisfactorily in reality, as in the shotgun example given earlier.

Unfortunately, we cannot say anything much stronger than that control architectures *must* be robust to the differences between the base set aspects of a simulation and the base set itself if they are to cross the reality gap. This is because there are so many ways of handling these differences depending on the behavior, what we demand of it, the nature of the robot-environment interactions and real-world features that make up the base set, and so on. As we shall see, however, there may be ways of forcing the *evolution* of this type of robustness (in whatever form evolution cares to conjure up), in which case there is no need to focus too hard on exact mechanisms by which control architectures may be robust, as this job may be left to the evolutionary process itself. All that is important for our present purposes is to say that this type of robustness is a necessary property of behaviors if they are to perform satisfactorily

in reality. Throughout the rest of the article, we will refer to controllers that display this property as being *base set robust*.

3 The Radical Envelope-of-Noise Hypothesis: A New Methodology for Building Simulations

In the previous section, two properties were described—that of being base set exclusive and that of being base set robust—which together are sufficient for the successful transfer of robot controllers across the reality gap. In this section, we introduce the radical envelope-of-noise hypothesis, a general methodology for forcing the evolution of these two properties by placing certain restrictions on the simulation itself. The section begins with a brief overview of the methodology so that the reader has a framework in which to view the arguments presented later. It then examines exactly how one might go about building simulations that force successful evolved behaviors to be both base set exclusive and base set robust. The section ends with a discussion of the issues involved in using the methodology to build minimal simulations for evolutionary robotics that run fast and are easy to build.

3.1 Overview of the methodology

As opposed to a single bold statement of fact, which may be proved true or false by empirical testing, the radical envelope-of-noise hypothesis should be seen as a methodology that we can use to build simulations that are posited to display certain properties. Whether or not the hypothesis is true (or, more pragmatically, useful) depends on whether reliably fit control architectures, that have evolved in simulations built using the methods and techniques outlined in the following sections, transfer across the reality gap. The methodology can be summed up in three steps:

1. A base set of robot-environment interactions (that are sufficient to underlie the behavior we want to evolve) must be identified and made explicit. A simulation should then be constructed that includes a model of these interactions. This simulation will display base set aspects, which have a basis in reality, and implementation aspects, which do not.
2. Every implementation aspect of the simulation must be randomly varied from trial to trial so that evolving controllers that depend on them are unreliable. In particular, *enough* variation must be included to ensure that evolving controllers cannot, in practice, be reliably fit unless they are base set exclusive (i.e., they actively ignore each implementation aspect entirely).
3. Every base set aspect of the simulation must be randomly varied from trial to trial so that reliably fit controllers are forced to be base set robust. The extent and character of this random variation must be sufficient to

ensure that reliably fit controllers are able to cope with the inevitable differences between the base set aspects and reality but not so large that reliably fit controllers fail to evolve at all.

Although only briefly stated, it should be evident that the methodology departs radically from more traditional ways of building simulations in steps 2 and 3. The next two subsections fully explain, in turn, the reasoning behind these two steps.

3.2 Ensuring that reliably fit controllers are base set exclusive

When control architectures are being evolved in simulation to perform a specific task, a fitness criterion is used (usually an explicit function tailored to the task) to tell which controllers are fitter than others. If this fitness criterion is set up correctly, then all control architectures that are able consistently and robustly to perform the task we are after will also be reliably fit and vice versa. If a single fitness evaluation consists of taking the average score from several independent trials, then we may say that a behavior that is reliably fit is one that scores a high fitness value on all such trials. In such a situation, a control architecture that is base set exclusive is one that uses the base set aspects of the simulation, and those aspects alone, to be reliably fit. If we want successful evolved behaviors to be base set exclusive, therefore, we need to ensure that reliably fit individuals do not depend in any way on implementation aspects of the simulation to achieve reliable high fitness, but depend only on base set aspects.

One way of forcing this to happen is to make all the implementation aspects of a simulation *unreliable* by varying them randomly from trial to trial. If this is done correctly, then the only practicable way in which a control architecture can be *reliably* fit, trial after trial, is by using those aspects of the simulation that are in themselves reliable (i.e., the base set aspects). Because a single fitness evaluation involves several independent trials, reliably fit individuals will score more highly, in the long run, over those that are less reliable, and we may expect them to be selected for by the evolutionary process. If the process succeeds, and reliably fit individuals evolve, then we can be confident that they will rely exclusively on the base set aspects of the simulation to perform their behavior and therefore will be base set exclusive.

The most difficult part of making the implementation aspects of a simulation unreliable is identifying what these aspects are in the first place. Mostly, they will arise as an incidental artefact of the modeling process, in which case they can be subtle and hard to spot. For example, one of the robot-environment interactions we might choose to include as a member of our base set is the fact that a particular sensor returns a value in the interval 0 to 13 when pointed in a certain direction. In implementing this interaction as a base set aspect of the simulation, however,

we must choose a particular *way in which* values are returned between 0 and 13. Values could be returned from across the whole interval, or they could all equal 7. The point is that unless the way in which values are returned within this interval in simulation is the same as the way in which they are returned in reality, then it is an implementation aspect of the simulation and has no real-world basis. If the way in which values are returned within the interval 0 to 13 is randomly varied from trial to trial, however, then evolving controllers cannot rely on *how* they arise within this interval (the implementation aspect), but only on the fact that they do (the base set aspect).

In other cases, the implementation aspects of a simulation will be obvious to us as we have put them in especially to make the modeling process easier or to reduce computational overheads. For example, we may note that certain of the robot-environment interactions we would like to include in our base set are very simple to model when the robot is located within certain areas of its environment and are very difficult to model in others. To make the job of building our simulation easier, therefore, we might model the real robot-environment interactions for when the robot is situated in the easy-to-model areas, and implement arbitrary interactions for when the robot is situated in the hard-to-model areas. In this case, the interactions between the virtual robot and its environment when it is situated in the easy-to-model areas are base set aspects of the simulation, and the interactions between the virtual robot and its environment when it is situated in the hard-to-model areas are implementation aspects of the simulation. By randomly varying these implementation aspects from trial to trial in a way that makes them completely unreliable, reliably fit controllers will employ strategies that rely on the robot-environment interactions in the easy-to-model areas, while completely ignoring any interactions between the robot and its environment in the hard to model areas. Extra care must be taken in this sort of situation to ensure that the base set aspects of the simulation are comprehensive enough to allow reliably fit controllers to evolve, however. There is a real danger, if we are overzealous in our lust for computational expediency, that we may effectively exclude so many real-world features from the simulation that what is left is insufficient for successful behavior.

Once the implementation aspects have been made explicit, we must then tackle the task of injecting randomness into the implementation aspects of the simulation. In many cases, it will be tempting merely to add large amounts of noise to everything that is not a base set aspect and to leave it at that. However, if this noise is in itself reliable in the sense that evolving controllers can always count on it being there, then they can and will (see Jakobi et al., 1995) evolve to use it to achieve high fitness. The secret is to vary the implementation aspects of the simulation randomly from trial to trial as opposed to during each trial only. As a fitness evaluation consists

of several trials, each controller will be subjected to several different instances of each implementation aspect—noisy, absolute, black, white, big, small or whatever—depending on the nature of the particular aspect and the ways in which it can be varied. As long as there is nothing that all instances of a particular implementation aspect have in common, then reliably fit controllers will be totally independent of that aspect, or they will not be reliable.

Of course, in practice, it might be very difficult to ensure that there is nothing that all instances of any particular implementation aspect have in common. However, if the implementation aspects of a simulation are made unreliable enough, then it is *so* much harder for evolution to find a way of using them reliably than it is for evolution to find a way of totally ignoring them that we can be extremely confident that controllers that evolve to be reliably fit will be base set exclusive.

3.3 Ensuring that reliably fit controllers are base set robust

To ensure that reliably fit control architectures are base set robust, we must be able to ensure that they can cope with the differences between the base set of robot-environment interactions on which a particular simulation is founded and the base set aspects of that simulation. We may approach this by adapting ideas borrowed from Husbands and Harvey (1992) (with further elaborations in Husbands, Harvey, and Cliff, 1993). The concept is that by randomly varying the base set aspects of a simulation by a small amount, from trial to trial, reliably fit individuals will have to be able to cope with a certain amount of variation in order to be reliable. There will, therefore, be a selection pressure in favor of controllers that are better able to cope with slightly different versions of each base set aspect and, thus, in favor of controllers that are better able to cope with the differences between the base set aspects of the simulation and the base set of robot-environment interactions in reality. Hence, to evolve reliably fit individuals that are base set robust, all we need is some way of knowing how much random variation must be applied to the base set aspects of the simulation and the best way in which to apply it.

As has already been said, rarely is it possible to simulate even the smallest portion of the world completely accurately. However, it also is rare that a simulation builder will not have at least some idea of how *inaccurate* his or her simulation is, and this seems a sensible way to work out limits on the amount of random variation we need to apply to the base set aspects of a simulation in order to ensure that successful evolved controllers will be base set robust.

As to how this variation should be applied, there are lessons to be learned from experiments reported in Jakobi et al. (1995). In those experiments, extra noise was added to the simulation over and above that present in reality, and controllers were able to evolve that made use of the extra noise in such a way that they were reliably fit

in simulation but failed miserably when downloaded onto the real robot. However, this extra noise was *reliably* present during every trial, so evolving controllers that relied on its presence were still able to be reliably fit. In other words, evolving controllers were faced with the same base set aspects of the simulation at every fitness trial, and these base set aspects were significantly different from the real base set of robot-environment interactions (i.e., they were much noisier). Given this fact, it is unsurprising that evolved controllers were unable to cross the reality gap as they had not evolved to be able to cope with lots of different instances of each base set aspect but only with a single instance that had no basis in reality.

For there to be a selection pressure in favor of controllers that can cope with slightly differing versions of each base set aspect of a simulation, these aspects need to be varied *between* trials and not during them. There should, of course, be noise on sensors and actuators during each trial, as there will also be noise on sensors and actuators in the real world. However, this noise should be regarded as an integral part of the base set of robot-environment interactions on which the simulation is founded and not something extra to it. Noise levels should be altered between trials along with all the other base set aspects of a simulation. They should not remain steady throughout the evolutionary process at unrealistic levels.

3.4 Minimal simulations

A careful inspection of the last two sections will reveal that nowhere is it implied that the base set aspects of a simulation should reflect reality as closely as possible, nor that the number of implementation aspects of a simulation should be kept to a bare minimum. In this lies the potential power of the radical envelope-of-noise hypothesis. A reliably fit controller that evolves in a simulation containing very inaccurate base set aspects and lots of implementation aspects is just as likely as any other to cross the reality gap, *provided* that the right amount of random variation is included in the simulation in the right way according to the methodology laid out earlier. What *is* much more unlikely in this situation is that reliably fit controllers will evolve at all. The amount of randomness with which the evolutionary machinery can find ways of coping will always be limited, no matter how this machinery is set up. If the amount of variation necessary to ensure that reliably fit controllers cross the reality gap surpasses this limit, then reliably fit controllers simply will fail to evolve.

However, if the evolutionary machinery *is* sufficiently powerful, we can evolve complex control architectures, capable of performing nontrivial real-world tasks, using surprisingly inaccurate and simple simulations. In such a situation, how one chooses the members of the base set of robot-environment interactions, and builds a model of them, may be governed in the main by considerations of computational expense rather than those of fidelity. In other words, if we are interested in evolving

a particular behavior, we can build the minimal base set of robot-environment interactions necessary into a model optimized for speed rather than accuracy. Using this, we then can construct a minimal simulation, which may include implementation aspects that lead to further run-time savings, that is very computationally efficient.

Minimalism has trade-offs. By providing only a minimally sufficient base set of robot-environment interactions within a minimal simulation, we are depriving evolution of its opportunistic ability, when performed within the real world, to ground reliable behavior in any aspect of the environment it sees fit, whether or not we have thought of it.[4] However, this point also has its positive side. By making explicit and modeling a minimal base set of *behaviorally relevant* robot-environment interactions, we ensure that the only controllers that can evolve to be reliably fit within the simulation will be those that are able to perform the "real" task in a general, nonbrittle way, without relying on non–behaviorally relevant aspects that are specific to the particular environment within which they evolved. Thus, the famous neural network, developed by the U.S. military, that could discern pictures of landscapes containing tanks from pictures of landscapes that did not—owing to the unfortunate fact that all the pictures containing tanks were taken in the morning whereas those that did not were taken in the afternoon—could not evolve in a minimal simulation.

Both of the experiments described next involve minimal simulations. In the first, a rough and ready simulation of a Khepera robot is used to evolve controllers that are capable of reliably solving a T-maze in the real world and, in the second, a simulation of the Sussex University gantry robot is used to evolve controllers that can robustly perform the task described in Harvey et al. (1994)—consistently telling a triangle apart from a square using real vision. The methodological guidelines proposed in this section are very general,[5] not a step-by-step recipe that, if used, will guarantee the user success every time. It is hoped, however, that together with the example simulations in sections 4 and 5, this information provides readers with at least some idea of how they might go about constructing good minimal simulations for evolutionary robotics.

A Minimal Simulation of a Khepera Robot

This section describes experiments in which neural network controllers were evolved for a Khepera robot using a minimal simulation. This robot, shown in Figure 1, is

4 Adrian Thompson's work with evolvable hardware (Thompson, 1996) provides a beautiful example of this type of artificial evolution.

5 Although the techniques proposed have been developed with evolutionary robotics firmly in mind, there seems no reason why they could not be adapted for use in some more conventional engineering domains that would also benefit from fast simulations that are easy to build.

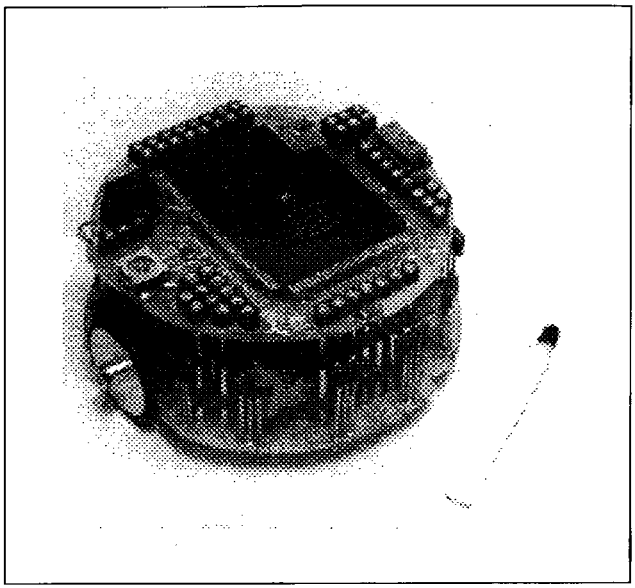


Figure 1
The Khepera robot.

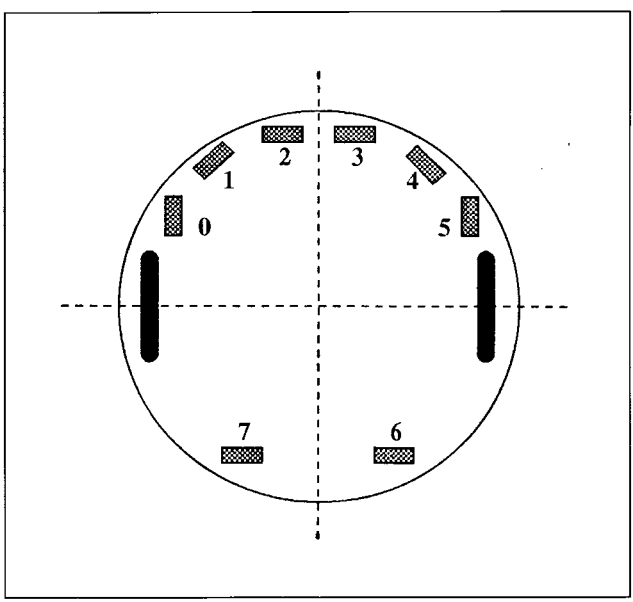


Figure 2
A plan of the Khepera robot showing the positions and numbers of the infrared sensors and the two wheels.

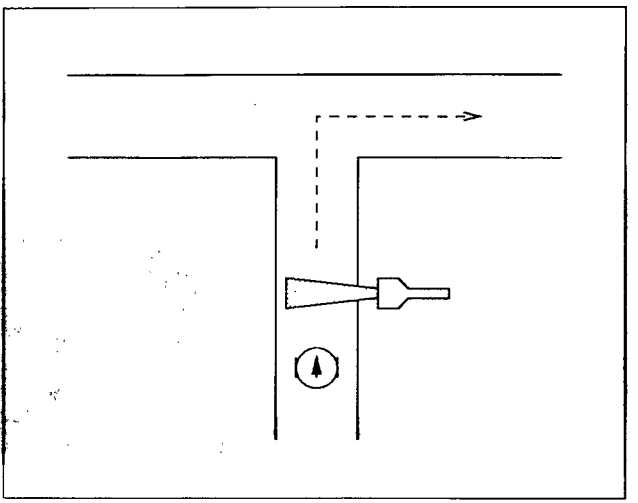


Figure 3
The task in the real world.

5.8 cm in diameter and approximately 3 cm high. Eight infrared sensors, which respond to nearby objects, are placed around the robot body as shown in Figure 2. In a different mode, these sensors may also be used to detect ambient light levels in the vicinity of the robot, with very rough directional sensitivity (see K-Team, 1993). Several different groups (Jakobi et al., 1995; Michel, 1995; Miglino et al., 1995) have built Khepera simulators on which they have successfully evolved control architectures that cross the reality gap. For this reason, the Khepera is an ideal platform on which to test the radical envelope-of-noise hypothesis.

4.1 The aim

The aim of the experiments was to evolve a behavior for the Khepera robot that was at least one step up from the simple reactive behaviors that have been prevalent in the evolutionary robotics literature thus far. The behavior that was selected is shown diagrammatically in Figure 3. As a Khepera robot begins to negotiate a T-maze, it passes through a beam of light shining from one of the two sides, chosen at random. To score maximum fitness points, the control architecture must "remember" on which side of the corridor the light went on and, on reaching the junction, must turn down the corresponding arm of the T-maze. This behavior involves several elements: Not only must controllers guide the robot down the corridors without touching the sides and negotiate the junction at the end of the first corridor (simple reactive behaviors both), but they also must involve a dependence on some internal or (less likely in this case) external state that allows them to "remember" which side the lamp was on so that they can take the correct turn at the junction.

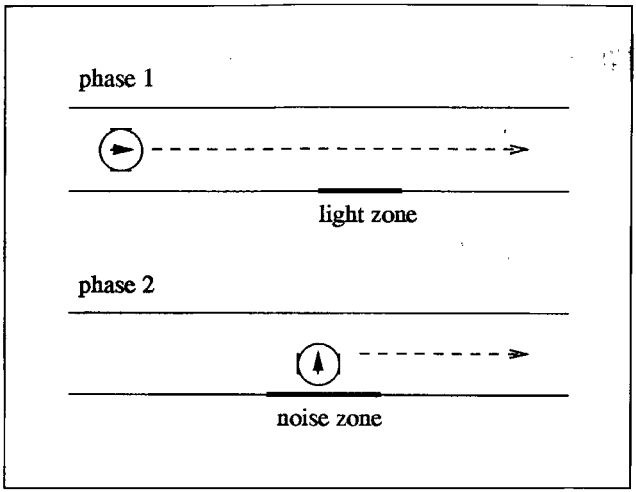


Figure 4
The T-maze task in simulation.

4.2 The minimal simulation

The minimal simulation used in the experiments was designed with low computational overheads firmly in mind. To give some idea of its simplicity, it contains two look-up tables, one containing 72 values and one containing 80, and approximately 300 lines of commented C code that employ nothing more mathematically complicated than floating point arithmetic. In fact, it does not model a T-maze at all—or rather it does not model all aspects of a T-maze—but only a sufficiently large base set of robot-environment interactions for the evolution of successful behaviors. This particular minimal base set was chosen because its members are easy to model; the robot-environment interactions in question are the same whether the robot is in a T-maze or in a simple, continuous, straight corridor and, as we shall see, this allowed considerable simplification of the simulation. The base set consisted of the following interactions:

1. The way in which the robot moves in response to motor signals
2. The way in which the infrared sensors return values in response to those sections of the walls of the T-maze that can be regarded as if they were sections of the walls of a continuous, infinite corridor
3. The way in which the ambient light sensors respond to bright verses ambient light levels

At first glance, item 2 of this list seems totally counterintuitive as a T-maze has nothing to do with infinite corridors. However, with respect to the infrared sensors of a Khepera robot, which have a maximum range of only 8 cm or so, a T-maze

is identical to an infinite corridor almost everywhere. Where a T-maze differs from a corridor, at the T-junction, the interactions between the sensors and the corridor walls were treated as implementation aspects of the simulation and were randomly varied from trial to trial according to the methodology laid out earlier. In this way, reliably fit controllers were forced to use strategies that depended on the interactions between the infrared sensors and the sections of the walls of the T-maze that could be regarded as straight and continuous corridor walls, and those interactions alone. First, we will describe the way in which the simulation of a T-maze was constructed from two different phases of a simple continuous corridor model, and then we will describe how the corridor model itself was put together.

4.2.1 Simulating a T-maze with two corridors Figure 4 shows the two phases of the T-maze simulation. In the first phase, the virtual robot had to travel down a simple corridor, where it received a light signal from either one side or the other. After it had traveled a predetermined distance, it suddenly was popped out of the first corridor, rotated through 90 degrees, and popped into the middle of a second corridor for phase 2. It then had to choose whether to turn left or right, depending on which side the light had been on, in order to gain maximum fitness points.

Now, although this twin corridor set-up varies significantly from a T-maze, the two have enough in common that evolving control architectures that are prohibited from relying on any of the differences still are able to sense enough of their environment to perform the task successfully. In particular, the robot-environment interactions governing the way in which the robot, traveling down a straight corridor, is confronted with a wall straight across its path and a second corridor stretching off to either side all were modeled.

The differences between the simulation and the real-world T-maze all occur around the T-junction. When the virtual Khepera robot suddenly appears in the second corridor facing the wall at the start of phase 2, there is a continuous wall directly behind it (see Fig. 4). In reality, when the Khepera is confronted with a wall across its path and is forced to make its decision about which way to turn, there is a complicated junction in the wall behind it (see Fig. 3). Because of this, the simulated robot's infrared sensor interactions with the simulated back wall, in an area corresponding to where the corridors meet in reality and approximately 5 cm to either side, were regarded as implementation aspects. If this section of the wall fell within range of the infrared sensors, then the way in which these sensors reacted varied randomly from trial to trial: Sometimes they returned maximum values, sometimes low values, and sometimes totally random values. In this way, reliably fit controllers were forced to employ strategies that, at the decision point, were oblivious to this

difference between the simulation and reality, relying only on the fact that there was a straight, continuous wall in front of them and space to either side.

4.2.2 Ensuring that reliably fit controllers were base set exclusive To ensure that reliably fit controllers were base set exclusive, all the implementation aspects of the simulation were identified and rendered unreliable. In addition to those implementation aspects concerned with the differences between the simulation and reality around the area of the T-junction, there were several others:

- The side of the corridor from which the light signal came
- The width of the two corridors: between 13 cm and 23 cm
- The exact starting orientation of the robot: between 22.5 and −22.5 degrees of facing straight down the corridor
- The length of the illuminated section of the corridor: between 2 cm and 12 cm
- The total length of the corridor in phase 1: between 40 cm and 60 cm

To these attributes of the simulation it was necessary to give values in order that the simulation be a consistent whole, but we did not want evolving behaviors to be able to rely on these attributes. Random values, from within the ranges shown, were assigned to each implementation aspect at the start of each trial. Reliably fit controllers therefore were forced to be independent of exactly where each value fell within the relevant range and thus were base set exclusive.

4.2.3 Simulating an infinite corridor A simple model of a Khepera's robot-environment interactions within an infinite corridor was responsible for generating the base set aspects of the simulation. At each iteration, two main functions were called: one that updated the virtual Khepera's position and one that calculated the values returned by the infrared sensors. The third robot-environment interaction listed previously, namely the way in which the ambient light sensors react to bright versus ambient light levels, actually was handled by a single line of code. We will look at the way all three robot-environment interactions were computed in turn.

The simulation was updated the equivalent of ten times per second. Figure 5 shows how the new position of the virtual Khepera robot within its environment was calculated at each iteration. The orientation was used as an index to a look-up table with 36 pairs of values: horizontal and vertical increments for a Khepera traveling at a speed of 1 cm/sec. To work out its new position, the values returned from this look-up table were multiplied by the average wheel speed in centimeters per second. The speed of each wheel was calculated directly from multiplying the

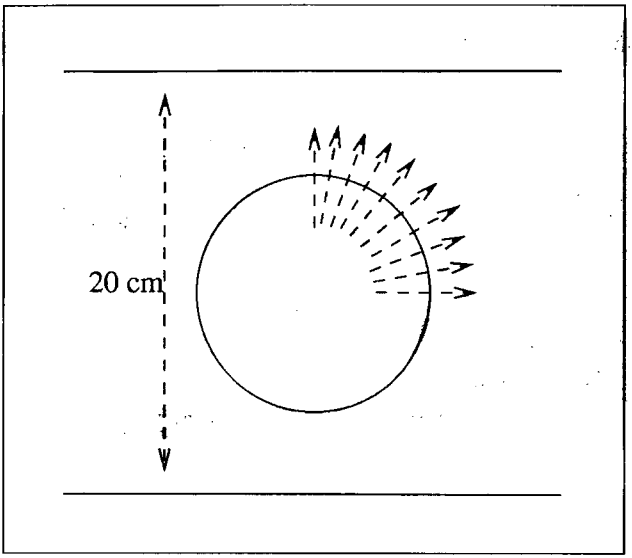


Figure 5
Diagram depicting how the new position of the Khepera robot was calculated at each position. A look-up table holds horizontal and vertical increment values for 36 different orientation values and an average speed of 1.

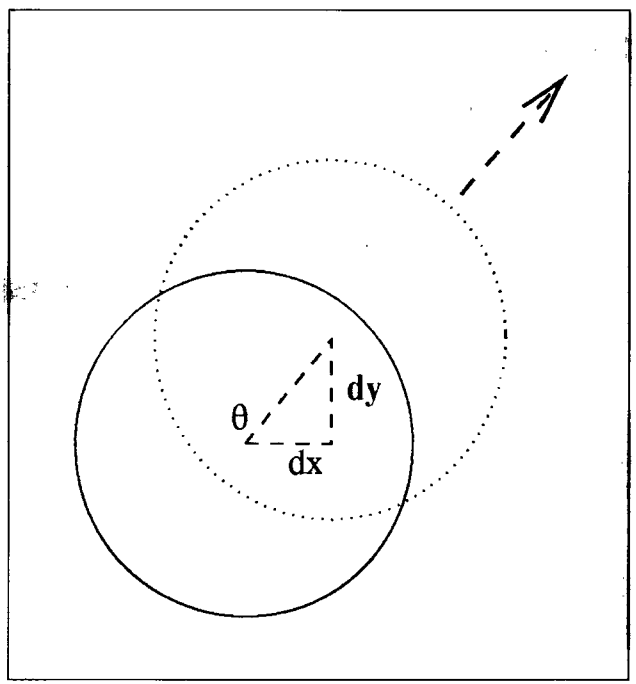


Figure 6
Features that formed the basis for the values in a look-up table containing the perpendicular distances to the walls of a 20-cm-wide corridor for all eight sensors in ten possible orientations.

motor signals by the constant 0.8 cm per motor unit per second. The change in orientation at each iteration was equal to the difference between the distances the two wheels moved divided by the radius of the robot (approximately 5.2 cm). There was no allowance for momentum, and the noise inherent in the real-world situation was not modeled.

Calculating the infrared values was a slightly more involved process and proceeded in three stages. First, the robot's orientation was used to generate rough distance-to-wall metrics for each sensor, as if the robot was in the center of a 20-cm-wide infinite corridor. Second, these values were scaled according to the actual width of the corridor in the simulation and the distance from the robot to each wall. Third, the scaled distance-to-wall metrics were used to calculate infrared sensor values by way of a simple linear relationship. This process is described in more detail later.

Figure 6 demonstrates these features on which the values held in the infrared look-up table were based. There were ten sets of eight values, each set corresponding to one of ten different robot orientations, from facing straight down the corridor to perpendicularly facing one of the walls. The values themselves were based on the distances from the center of the robot (which is 10 cm away from each wall), along the lines of the corresponding sensors, to the walls of an infinitely long corridor. However, these distance values were in fact always slightly shorter than the line-of-sight distance to the wall in order to account (in a very approximate way) for the fact that the infrared sensors of a Khepera robot are sensitive over a whole arc rather than just along the direct line of sight extending out from each sensor (K-Team, 1993). If the distance from the center of the robot, along the line of a sensor, to a wall of the corridor were d, then the warped distance value wdv stored in the look-up table was given by the following equation:

$$wdv = 10 + (d - 10)/3$$

The constant 3 in the denominator was chosen fairly arbitrarily and without accurate measurement purely on the basis that it gives the equation roughly the right properties.

The minimum possible value stored in the table, therefore, for a sensor directly facing the wall, was 10 cm. The maximum possible value, for a sensor directly facing down the corridor, was infinity. Now, although there were only ten sets of values stored in the table (one set of eight for each multiple of 10 degrees between 0 and 90 degrees inclusive), it was a simple matter to calculate sets of values for any other multiple of 10 degrees. These calculations were made by taking the particular orientation angle in question and rotating it by the appropriate multiple of 90 degrees until it lay in the correct quadrant. The look-up table then was used to ascertain a set of values, and these were reflected across the midline of the robot if necessary

(i.e., if the angle was between 90 and 180 or between 270 and 360). If the robot was in the center of a 20-cm-wide, infinitely long corridor, therefore, warped distance values could be calculated for any sensor at any orientation.

In practice, the perpendicular distance from the center of the robot to a particular wall of the corridor was variable. Values were scaled appropriately, however, by multiplying all values returned from the look-up table (for sensors that pointed at that particular wall) by the fraction attained by dividing the actual distance to the wall by 10 cm. For example, if the distance from the robot to a wall was actually 5 cm instead of 10 cm, then look-up-table values for sensors that pointed at that wall were halved. In this way, the 80 values of the look-up table were sufficient to find the approximate distance, warped according to the equation given earlier, from the center of the robot in any position and any orientation, along the line of any sensor, to the wall of an infinite corridor of any width.

Having ascertained warped distance values (wdv) for each sensor, the actual value that each simulated sensor returned, V, was given by a simple linear function:

$$V = \begin{cases} 0 & wdv > a \\ 1024 \times (7 - wdv)/2 & a > wdv > b \\ 1024 & b > wdv \end{cases} \quad (1)$$

where a and b were the maximum and minimum extent, respectively, of the linear part of the response function. This meant that a sensor would saturate at maximum value if its warped distance value were less than b (typically approximately 5), would return zero if its warped distance value were greater than a (typically approximately 9), and would respond linearly in between.

A simple random number generator was used to generate uniformly distributed random deviates in the range ± 50. These were added to returned sensor values at each iteration. In addition, the lowest value an infrared sensor could return was a random background value between 0 and 20. These noise levels roughly approximate the levels observed in the real world and, as such, were as much a part of the robot-environment interaction model as was any other aspect.

The way in which ambient light sensors respond to bright versus ambient light levels was modeled by a single line of code. When the robot entered a particular section of the corridor in phase 1 (that was randomly predefined in terms of length and position relative to the starting point), the values returned by the ambient light sensors on one side of the robot dropped from their normal background value of approximately 450 to a value of nearly 100, as if they had been illuminated by a bright light. When the robot left the special light zone, the values returned to their background levels. Whether the right side of the robot or the left side was illuminated depended on which side of the corridor the light source was placed and in which

of the two directions directly down the corridor the robot was closest to pointing. Random deviates in the range of ±50 were added to each ambient light sensor value at each iteration.

4.2.4 Ensuring that reliably fit behaviors were base set robust According to the methodology laid out in section 3, the base set aspects of a simulation must themselves be varied slightly from trial to trial in order to ensure that reliably fit controllers are robust to the differences between the model and the real world. This variation was accomplished in two ways in the simulation being discussed. First, random offsets of between ±1 cm/sec were generated at the beginning of each trial and were added to wheel speeds during position update calculations. This caused the virtual robot to move in a randomly predefined curve when it would normally have gone in a straight line, and thus forced controllers to devise ways of coping with and being robust to a variety of different movement characteristics. Second, the constants a and b of Equation 1 were set randomly at the beginning of each trial, the former in the range 7.1 to 10.1 and the latter in the range 4.1 to 6.1. This had the effect of forcing reliably fit controllers to devise strategies that were robust to a range of infrared sensor characteristics.

4.2.5 Summary of the simulation The overall shape of the simulation originated from the observation that a T-maze is like an infinite, continuous corridor (from the point of view of a Khepera robot's all-important infrared sensors) almost everywhere. Bearing this in mind, we constructed a simulation of a T-maze in which the base set aspects could be generated using a simple model of a Khepera robot's interactions with the walls of a continuous, infinite corridor. The nature of these modeled interactions was varied slightly and randomly, from trial to trial, to ensure that reliably fit controllers were base set robust. Those aspects of the simulation that corresponded to sections of the T-maze that could not be modeled by a continuous corridor were regarded as implementation aspects of the simulation. All implementation aspects of the simulation were varied randomly from trial to trial to ensure that evolving controllers were base set exclusive.

4.3 The evolutionary machinery

The experiments described here were designed to test the radical envelope-of-noise hypothesis (i.e., whether control architectures that have evolved to perform reliably a task in a simulation created according to the methodology outlined in section 3 would transfer successfully to reality). However, for the T-maze task described earlier, it is no simple matter to evolve reliably fit controllers in simulation in the first place. For this reason, the evolutionary machinery is described briefly here.

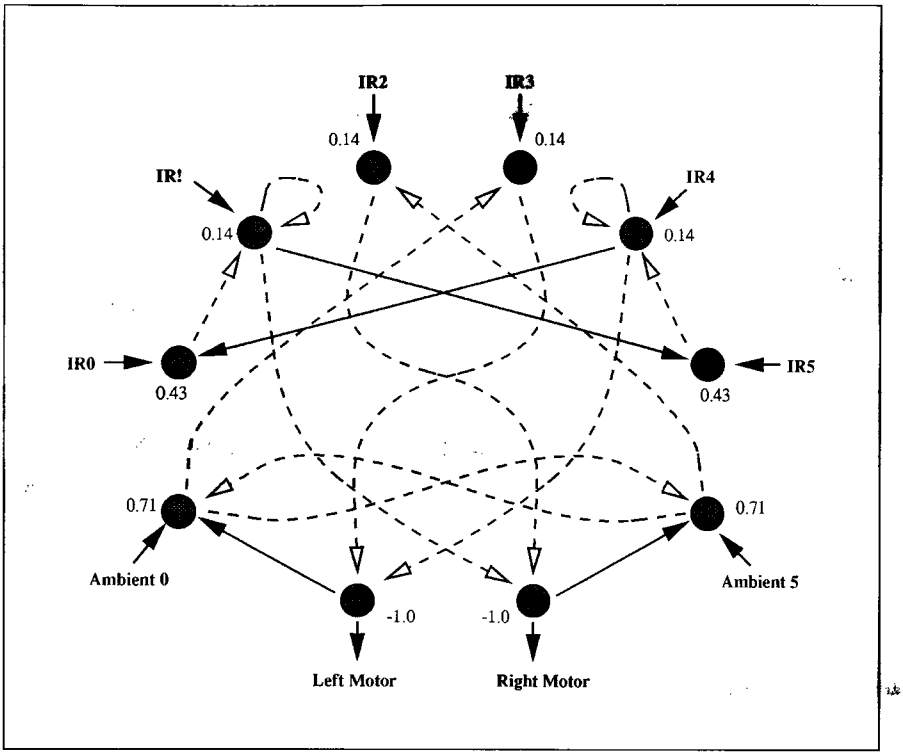


Figure 7
A typical evolved network. The solid arrows are excitatory links and the dashed arrows are inhibitory links; exact weight values are not shown. Threshold values appear next to each neuron.

The controllers themselves were arbitrarily recurrent neural networks. The number of neurons in a network was fixed for each evolutionary run (usually ten neurons), but all the links to a neuron from other neurons (up to a maximum of three) were genetically determined. Networks were forced to be bilaterally symmetrical by effectively evolving only half the network and reflecting it across the midline.[6] All nonmotor neurons had simple step threshold activation functions of the following form:

$$A_j = \begin{cases} 0 & \sum A_i w_{ij} < T_j \\ 1 & \sum A_i w_{ij} \geq T_j \end{cases} \quad (2)$$

where A_j is the activation of the jth neuron, T_j is the threshold of the jth neuron, and w_{ij} is the weight on the connection from the ith neuron to the jth neuron.

6. For a justification of why symmetry was enforced rather than allowed to evolve, see Jakobi (1996).

The activations of motor neurons were calculated using a slightly different output function:

$$A_j = \begin{cases} 0 & \sum A_i w_{ij} < T_j - 1 \\ \sum A_i w_{ij} - T_j & T_j - 1 \leq \sum A_i w_{ij} \leq T_j + 1 \\ 1 & \sum A_i w_{ij} > T_j + 1 \end{cases}$$

Thresholds were real numbers in the range ± 1.0, and weights on links were real numbers in the range of ± 2.0. Figure 7, a diagram of a typical evolved neural network, shows how sensor value inputs were applied to networks and how motor values were output. All sensor values were normalized in the range 0 to 1, and motor outputs were multiplied by a factor of 10 to give motor signals in the range ± 8 cm/sec. The network, sensor values, and motor outputs (in fact, the entire simulation) were updated the equivalent of ten times per second.

A direct-encoding scheme was used; there was a one-to-one mapping between genotype and phenotype. Each genotype was a string of 140 bits, consisting of five fields or genes, one for each neuron of the left-hand side of the network. The neurons on the right-hand side of the network were the exact mirror image of those on the left-hand side. Each gene was itself divided into fields. The first four bits of each gene, a binary number between 0 and 16, defined the threshold of that neuron by normalizing between ± 1. The next three sets of eight bits defined the three possible links to that neuron from other neurons in the network: the first four ascribing one of 16 possible values to the weight of the link between ± 2 and the next four bits defining from which of 16 neurons the link came. Because there were only ten neurons in the network, if a link indexed a nonexistent neuron, then it did not connect, thus placing under genetic control the number of links to a neuron.

The fitness function returned the average value scored by an individual in a total of ten fitness trials, each lasting the equivalent of 15 seconds. At the end of each trial, the fitness value was equal to the total distance traveled through the corridor system plus a bonus of 100 if the virtual robot went the right way at the T-junction. Thus, if the virtual robot traveled a distance d_1 in the first corridor and a distance d_2 at the second corridor, then the fitness score T for that particular trial was calculated by:

$$T = \begin{cases} d_1 + d_2 + 100 & \textit{right way at lights} \\ d_1 + d_2 & \textit{wrong way at lights} \end{cases}$$

The genetic algorithm was a steady-state distributed genetic algorithm (Collins & Jefferson, 1991) with a population of 100 individuals arranged on a virtual 10×10 grid. At each iteration, a random location was chosen on the grid and a breeding

pool constructed from the nine individuals of the 3×3 square centered on that location. Two probabilistically fit parents were chosen from this breeding pool according to a linear rank-based selection procedure, and an offspring was constructed by a process of crossover and mutation. This offspring then replaced a probabilistically unfit member of the same breeding pool according to an inverse linear rank-based selection procedure. Single-point crossover was applied with probability 0.7, and the expected number of bitwise mutations per genotype, according to a Poisson distribution, was 2. At each offspring event, not only was the offspring's fitness evaluated, but the fitnesses of both parents also were reevaluated in an attempt to cope with the noise inherent in the evaluation process.

4.4 Experimental results

Figure 7 shows a typical example of the sort of neural network that consistently evolved within approximately 1000 generations (where a generation was taken to be 100 offspring events). This is the simulated equivalent of $300 \times 15 \times 10 \times 100 = 45{,}000{,}000$ seconds, or more than 17 months of continuous real-world evolution, and takes approximately 4 hours to run as a single user on a SPARC Ultra (Sun Microsystems, California). The network reliably achieved near-optimal fitness within the simulation. To determine whether the network would transfer successfully across the reality gap, the network was downloaded onto a Khepera robot, and its ability to perform the task in the real world was tested. Sixty different trials were performed one after another, 20 in each of three different widths of corridors, the light being on the left for 10 trials and on the right for the other 10. The consequent robot behaviors were filmed from above so that the exact path taken by the Khepera robot on each trial could be extracted using basic image-processing techniques and overlaid on aerial views of the setup. The results of this process are the six images in Figure 8.

In the top pair of images, the corridor is only 11 cm wide, and the paths taken by the Khepera robot on all 20 occasions are tightly constrained. In the second pair of images, where the corridor is 18 cm wide, and especially in the bottom pair of images, where the corridor is 23 cm wide, the paths taken by the Khepera robot are less constrained. Nonetheless, the Khepera still turns the correct way at the T-junction, even though on several occasions it must turn through greater than 90 degrees in order to accomplish this. Note that the path taken in most cases was near-optimal and that in every case the task was performed satisfactorily, which satisfied the criteria decided in section 2 for a control architecture to transfer successfully from simulation to reality.

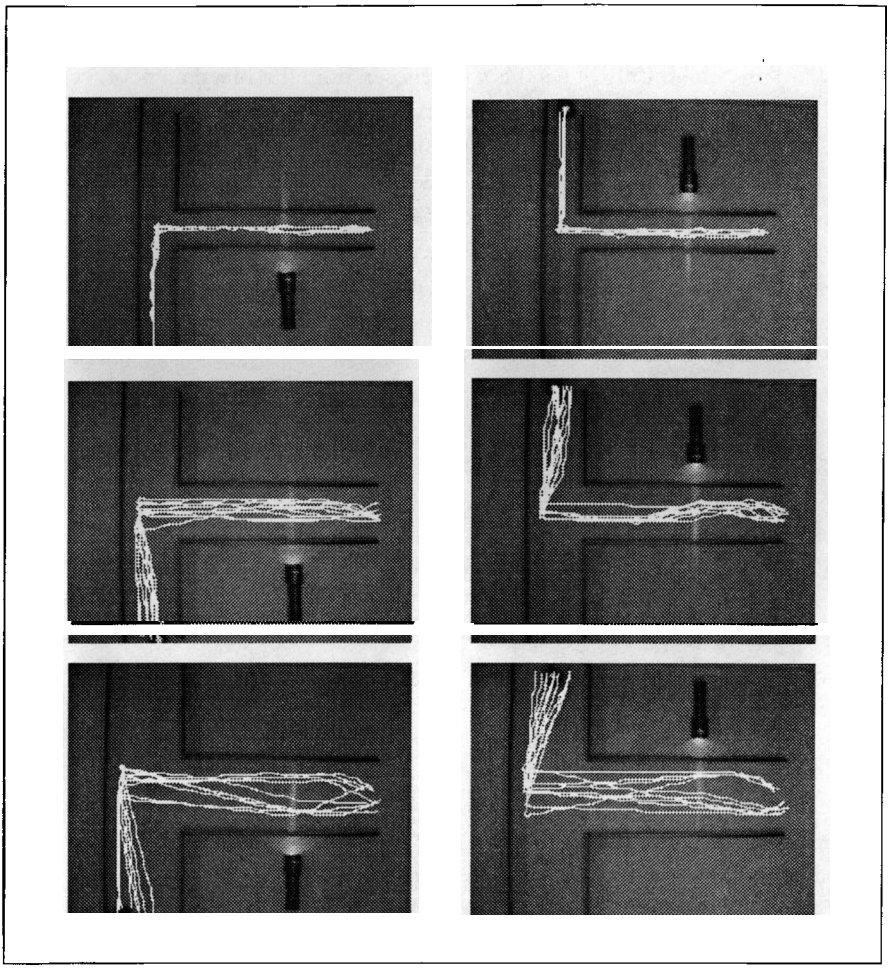


Figure 8
These six pictures together show the paths taken by a Khepera robot in 60 consecutive trials of the control architecture shown in Figure 7. These 60 trials were performed in consecutive batches of 10, and each picture shows 10 trials for a particular corridor width and torch orientation. The pictures were created using an overhead camera, a videodisc, and simple computer vision techniques to find the position of the robot in each frame.

5 A minimal simulation of the gantry robot

In this section, we describe experiments in which neural network controllers were evolved for the gantry robot. The gantry, shown in Figure 9, was developed at Sussex University for research into the evolution of visually guided behaviors. It

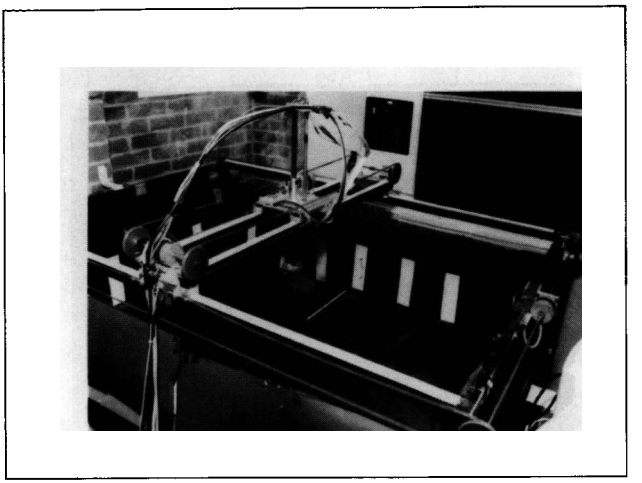


Figure 9
The gantry arena, with obstacles.

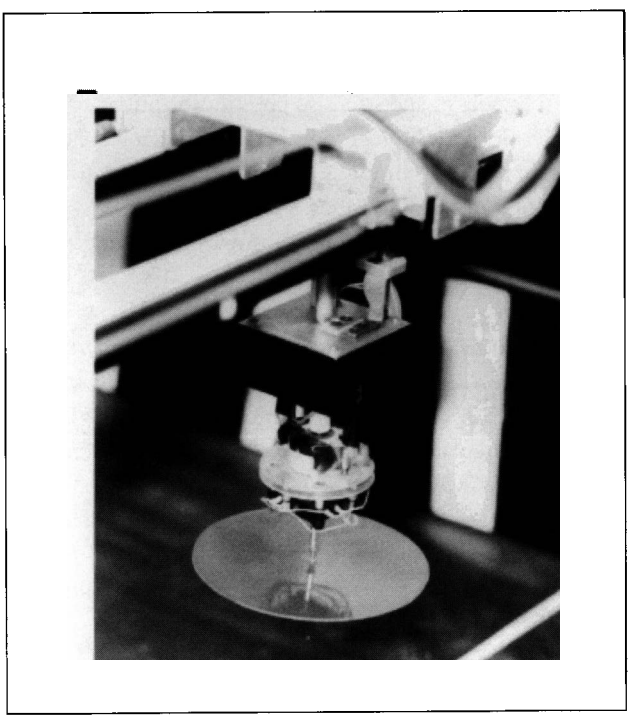


Figure 10
A close-up of the gantry robot.

is best thought of as a hardware simulation of a small, wheeled, mobile robot on top of which a camera is placed, that has been specifically designed so that control architectures can be tested automatically and safely in a highly controlled manner (Husbands, Harvey, Jakobi, Thompson, & Cliff, 1997).

Figure 10 shows a close-up of the actual robot. A camera points vertically downward at a 45-degree-inclined mirror to return a view from the robot looking straight out horizontally at the environment. The mirror is attached to a stepper motor that enables the mirror to rotate around the vertical axis under computer control and a dedicated vision personal computer then rotates the image array via software so that downward in the picture corresponds to downward in reality. The image array available for use by evolving control architectures therefore is equivalent to that produced by a camera pointing outward along the horizontal component of the mirror's orientation. The frame of the gantry robot is connected to two additional stepper motors that together allow the entire robot assembly to move in any horizontal direction within a rectangular arena (see Fig. 9).

All three stepper motors are controlled by a single-board computer (SBC) that is controlled, in turn, by a dedicated brain personal computer running the control architecture software. The brain computer sends commands to the SBC in the form of left and right wheel speeds, as if the gantry were a wheeled mobile robot. The SBC then calculates and issues stepper motor pulses so that the gantry robot moves in the appropriate fashion. From the point of view of control architectures running on the brain personal computer, therefore, the gantry robot behaves exactly as would a small, wheeled, mobile robot, controlled via the SBC, on top of which is set a camera whose image is accessed via the vision computer.

5.1 The aim

In Harvey et al. (1994), the authors report experiments in which both neural network control architectures and the visual morphologies of their inputs were evolved side by side to perform a simple shape discrimination task. Evolution occurred within an all-black rectangular arena, 150 × 100 cm, with 22.5-cm-high walls. Stuck onto one of the long walls were a near-square (20 cm wide × 22.5 cm high) and an equilateral triangle (20 cm wide × 22.5 cm high), both of which were cut from white paper. Starting from several different positions and orientations, evolving individuals were tested as to their ability to make the gantry robot move toward the triangle as opposed to the square (Fig. 11). After several generations, which took approximately 36 hours to perform in the real world, control architectures evolved that were able to perform the task. These controllers were approximately 80 percent reliable within certain constrained sets of lighting conditions (P. Husbands, personal communication, 1997): If the blinds of the laboratory were opened during the day, or if the overhead lighting

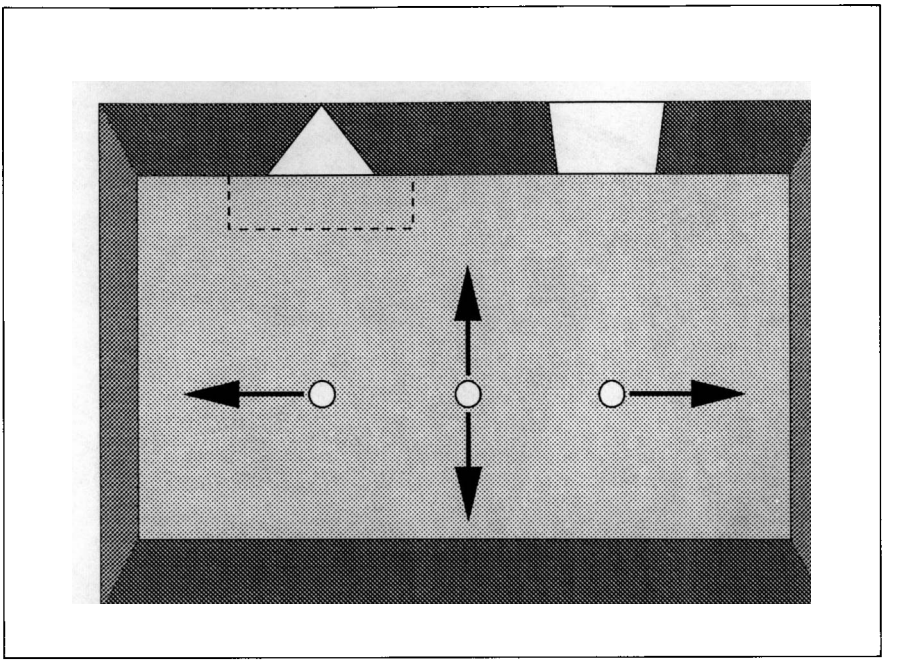


Figure 11
A diagrammatic view of the gantry arena from above, showing the four possible starting positions of the gantry robot. The dashed line in front of the triangle marks the area that the gantry must reach in order for a trial to count as a success for testing purposes.

was not on in the right way, the controllers failed. To remedy this sensitivity to differing lighting conditions, a set of lamps were strung up above the gantry robot, each turning on and off at different frequencies, to provide extreme real-world noise with which evolving controllers had to cope. The previously fit controllers failed completely when the "disco lights," as they are known at Sussex, were switched on. As yet, no new controllers have been evolved on the gantry robot using real-world evolution that are able to cope with the extra uncertainty provided by these lights. Therefore, it was decided that evolving reliably fit control architectures in a simulation built according to the methodology laid out in section 3, and determining whether these control architectures were able to perform the task satisfactorily in the real-world environment with the disco lights switched on, would provide a good test of the radical envelope-of-noise hypothesis.

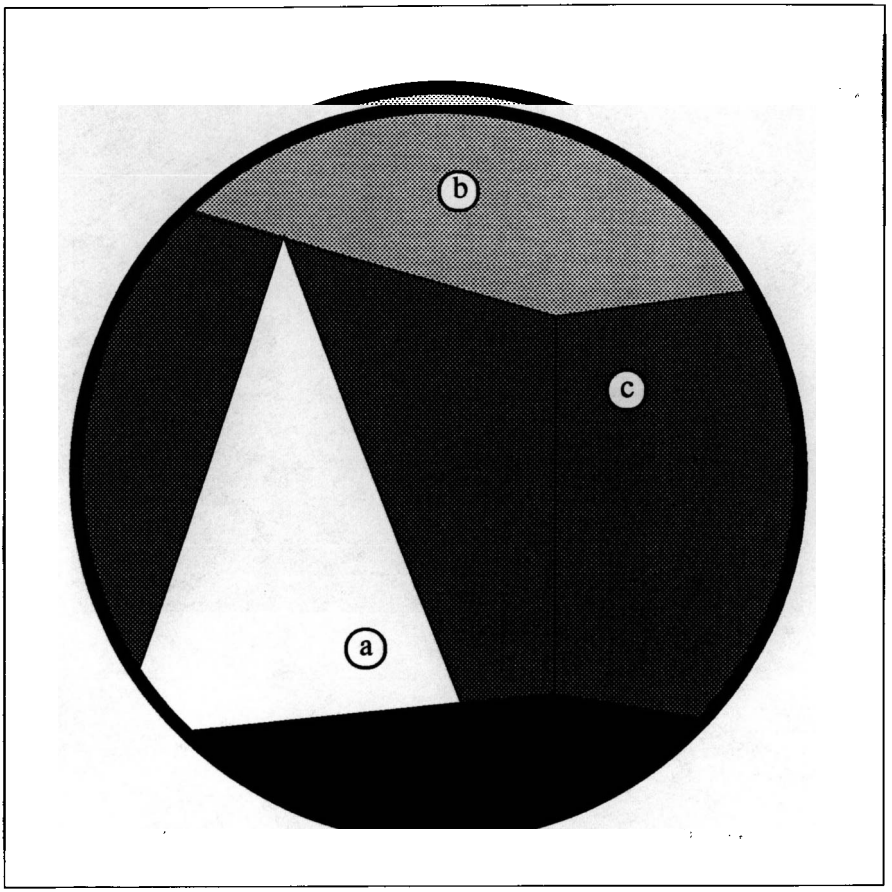


Figure 12
A typical image returned by the camera of the gantry robot. The robot is facing the corner of the arena and the triangle can be seen on the left. The white circles labeled *a*, *b*, and *c* are examples of pixels that project onto the triangle, ceiling, and wall, respectively. Pixel *a* will return a value between 14 and 15, pixel *b* will return a value between 0 and 15, and pixel *c* will return a value between 0 and 13. In the experiments reported in section 5, each visual input was made up of exactly one pixel the coordinates within the camera image of which were genetically determined.

5.2 The minimal simulation

In the experiments reported in Harvey et al. (1994), both the neural network control architectures and the morphology of their visual inputs were genetically determined. In the simulation experiments reported here, a different type of control architecture was used (see later), although both neural networks and the visual morphology of

their inputs again were genetically determined. The main difference between the two, as far as a simulation is concerned, is that in Harvey et al. (1994), each visual input to the neural network consisted of the average gray-level value of a genetically specified circular subregion of the camera image, whereas in the experiments reported here, each visual input consisted of the gray-level value of exactly one genetically specified pixel of the camera image (Fig. 12). In fact, these are not so different with respect to a simulation, as the average value of each circular visual field in Harvey et al. (1994) was just the average value of 25 randomly sampled pixels from within the field. A simulation of either, therefore, must contain a model of how specific pixels of the camera image acquire values in response to the orientation and position of the robot within its environment.

Under the disco lights suspended above the gantry, the values returned by pixels of the camera image vary widely with respect to both time and the direction of the camera. Even if we know the exact location within the arena onto which a particular pixel projects, there is little we can say about exactly what the value of that pixel will be. However, a few general things that hold true except in certain special circumstances: If a pixel projects onto a wall but not onto a shape, then it will return a value within the range of 0 to 13; if a pixel projects onto either the triangle or the square, then it will return a value between 14 and 15; and if a pixel projects onto either the floor or the ceiling of the arena, it will return a value between 0 and 15. Because these facts about pixel values within the disco-light environment are almost always in effect, and because they are enough to distinguish the white triangle and square from the black walls of the arena (for those pixels that project onto a wall of the arena), they are all that we needed to model.

In fact, it was logically essential to include in the model only one visual aspect in order that evolving control architectures would be able to perform the shape discrimination task; this aspect was the way in which pixels that project *onto the walls of the arena* acquire gray-scale values in response to the orientation and position of the robot. If a pixel projects onto the floor or ceiling, the value it returns will have nothing to do with squares or triangles, so there is no point in allowing evolving control architectures to rely on it. This is especially true when one considers the extra modeling required. For example, if the strategy employed by a control architecture that is reliably fit within the simulation depends on a pixel that projects onto the floor, then the simulated value of that pixel would have to be reasonably true to life, or the control architecture would fail when downloaded into reality: It would have evolved to rely on something that was true of the simulation but not true of the real world. For this reason, the values returned by pixels that projected onto the floor or ceiling of the arena were treated as implementation aspects of the simulation.

The base set of robot-environment interactions on which the simulation was founded, therefore, had just two members:

1. The way in which pixels of the camera image, that project onto the walls of the arena, return gray-scale values within certain intervals: 14 to 15 for pixels that project onto either the triangle or the square, and 0 to 13 for pixels that project onto the walls of the arena (but not onto either the triangle or the square)
2. The way in which the robot moves in response to motor signals

The model of the way in which the gantry robot moves in response to motor signals was adapted from the movement model for the Khepera robot explained in section 4. The simulation again was updated at a rate equivalent to ten times per second, and the same look-up table was used but with different constants to update speed, orientation, and position variables at each iteration of the simulation. The radius of the virtual robot (of which the gantry robot is a hardware simulation) is 15 cm, and the constant multiplied by the motor signals to give the current speed of the robot is 4.17 cm per motor unit per second. In addition, there was also a momentum term, m, such that at each iteration, the increment δv to each wheel speed v in terms of the required wheel speed u was:

$$\delta v = \frac{u - v}{m} \tag{3}$$

This momentum term was added for the simple reason that in the case of the gantry, robot momentum plays a significant role as it is a heavy robot that takes time to slow down and speed up. In the case of the Khepera, the robot is small enough and light enough that momentum effects can be regarded as modeling inaccuracies and can be coped with by reliably fit control architectures that are base set robust (see section 3.3). At every iteration, a random deviate in the range of ± 0.2 cm/sec was added to each wheel speed to approximate the noise inherent in the way the gantry robot moves.

Simple trigonometry was used to work out the location in the arena onto which a particular genetically specified pixel projects. Look-up tables were employed in place of the computationally expensive standard C library functions of *cos*, *sin*, and *tan*. Each table contained 360 values covering 360 degrees. In addition, there was a finer-grained look-up table for *tan* that contained 200 values, one for every 0.1 degree between 0 and 20 degrees.

After rotation by the vision personal computer, the image available to evolving control architectures on the gantry robot is a circular portion of a two-dimensional pixel array, 40 pixels in diameter and with an angle of acceptance of approximately

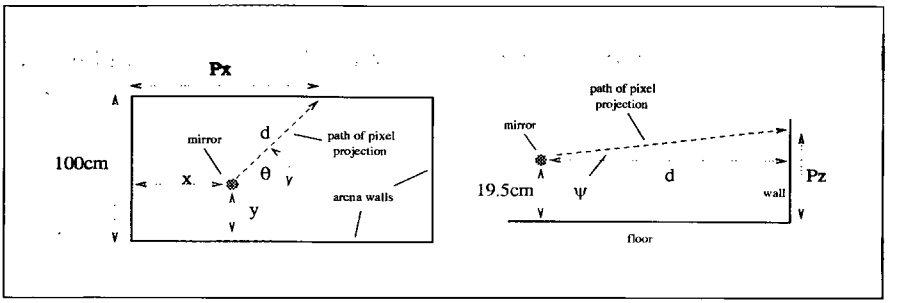


Figure 13
The left-hand picture shows the gantry arena as seen from above; if the horizontal angle at which a pixel projects from the mirror onto the back wall is θ, then $Px = x + \frac{100-y}{tan\theta}$ and $d = \frac{100-y}{sin\theta}$. The right-hand picture shows a cross-section of the gantry arena; if the vertical angle at which a pixel projects from the mirror onto a wall is ψ, then $Pz = d \times tan\psi + 19.5$.

50 degrees (see Fig. 12). The horizontal and vertical angular offsets of any particular pixel from the orientation of the robot were calculated from its x and y coordinates within the image and then were used to work out the horizontal and vertical angles at which the pixel projected out from the gantry robot's mirror relative to the fixed arena environment. Because the coordinates of the robot's position within the arena always were known, and the height of the mirror above the floor of the arena was fixed (approximately 19.5 cm), the exact horizontal and vertical coordinates of the spot onto which any particular pixel projected could be easily worked out. First, the simulation established which of the four walls of the arena a particular pixel would project onto if the vertical angle was in the correct range and calculated the horizontal coordinate of the pixel projection onto that wall. Second, the vertical coordinate of the pixel projection onto the wall was calculated. The way this was achieved is demonstrated in Figure 13. For calculations of Px (the horizontal coordinate of the point onto which a pixel projects), the coarse-grained *tan* look-up table was used and, for calculations of Pz (the vertical coordinate of the point onto which a pixel projects), the fine-grained *tan* look-up table was used. This is because ψ will always be a small angle somewhere between 0 and 25 degrees, whereas θ can be anything between 0 and 360 degrees.

Having worked out Px, Pz, and the relevant arena wall, the actual value attributed to a particular pixel depended on one of three possible scenarios: The pixel did not project onto a wall, in which case it returned a totally unreliable value that varied from trial to trial; or it projected onto a wall but not onto the triangle or square, in which case it returned a value between 0 and 13; or it projected onto the triangle or square, in which case it returned a value between 14 and 15. The *ways in which* values were retuned from within these intervals are described later. If Pz was less than 0 cm

or greater than 22.5 cm, then the pixel was judged to have projected onto either the ceiling or the floor. If *Pz* was between 0 and 22.5 cm, it was judged to have projected onto a wall. If the wall in question was the one on which hung the triangle and the square, then simple geometrical relationships between the coordinates of the pixel projection point and the vertices of the two shapes were used to determine whether the pixel projection point lay inside either of the shapes. At every iteration, a random deviate in the range of ± 1.2 gray-scale units was added to each pixel value.

5.2.1 Ensuring that reliably fit controllers were base set exclusive As reported earlier, if a pixel projected onto a wall but not onto a shape, then it returned a value between 0 and 13, and if it projected onto either the triangle or the square, then it returned a value between 14 and 15. Exactly how this was done is crucial, however, as the base set of robot-environment interactions did not include the *way in which* values are returned between 0 and 13 for black walls and between 14 and 15 for white shapes, but only the fact that they are. For this reason, the way in which pixel values are returned within these intervals was treated as an implementation aspect of the simulation and was varied from trial to trial according to the methodology outlined in section 3. This ensured that control architectures that had evolved to be reliably fit within the simulation worked independently of the way in which actual pixel values arose—as long as they arose within the specified intervals—and therefore that they were robust to the disco lights.

At the beginning of each trial, one of three ways of generating pixel values within the appropriate intervals was chosen:

1. Each pixel returned a different random value within the appropriate interval, and values varied randomly over time. This meant that whatever the behavior of the robot, values could change. The average time interval between changes in value for any particular pixel was taken from a Poisson distribution with an average length of 2 simulated seconds.
2. Each pixel returned a different random value within the appropriate interval, and values changed as a function of the robot's orientation. This meant that if the robot proceeded in a straight line, or remained still, then pixel values remained steady. If the robot turned, then pixel values could change. Values for each orientation were set randomly at the start of each trial, with angular distances between changes in value (for any particular pixel) uniformly distributed between 0 and 50 degrees.
3. Each pixel returned the same value, randomly set at the start of each trial, within the appropriate interval. Values for each interval were kept constant throughout the trial.

Pixels that projected onto either the ceiling or the floor were treated in a similar fashion, which ensured that they were totally unreliable: Random values were returned in a random way between the minimum and maximum values (0 and 15) instead of some subinterval. In this situation, reliably fit control architectures were not even able to depend on the interval within which returned values would lie.

The other implementation aspects on which reliably fit controllers were prevented from depending were the starting position of the robot at the beginning of each trial and the position of the triangle relative to the square (left or right) and vice versa. There were four possible starting positions that varied from trial to trial (see Fig. 11), and in half the trials that made up the fitness test, the triangle was on the left, whereas in the other half it was on the right.

5.2.2 Ensuring that reliably fit controllers were base set robust Various aspects of the model were varied from trial to trial in order to ensure that reliably fit control architectures were base set robust (see section 3.3). This was especially important with a robot such as the gantry, which is extremely noisy and imprecise in its operation. In particular, the mirror that reflects the horizontal image up into the camera is not set at exactly 45 degrees and is slightly warped. This means that objects appear differently depending on where they are in the camera image and that, as the robot approaches an object, its image will deform and distort, appearing to move upward. Because of this:

- A vertical angular offset of between -1 and -8 degrees was produced at the beginning of each trial. This was then added to the vertical angle of projection of every pixel throughout the trial.
- A horizontal angular offset of between ± 3 degrees was produced at the beginning of each trial. This then was added to the horizontal angle of projection of every pixel throughout the trial.
- The horizontal coordinates (with respect to the wall) of the four corners of the square and the three corners of the triangle each were offset by a random amount within the range ± 5 cm throughout each trial.

The stepper motors move the gantry supporting the robot along rollers, using drive-chains. The rollers slide rather than roll along their rails (owing to a design fault), with more friction in some places than others, and the drive belts are loose so that rapid sequences of motor commands can get lost in the extra "slop." Because of this, the robot can only approximate traveling at a constant speed and neither accelerates nor brakes evenly in response to motor commands. The robot often will cease completely half-way through a run. In order that reliably fit individuals evolved

to cope with these problems, (1) the momentum term, m, of Equation 3 was set randomly at the beginning of each trial to a value between 1 and 4, and (2) random offsets of between ± 0.5 cm/sec were generated at the beginning of each trial and added to required wheel speeds during position update calculations. Together these random variations ensured that reliably fit control architectures were able to cope with a wide variety of slightly different robot-environment interaction models. Included in this range were models that involved misshapen and malaligned mirrors as well as noisy and unpredictable motors—such as the model instantiated by the real gantry robot.

5.3 The evolutionary machinery

Evolving control architectures that visually discriminate between triangles and squares in a noisy real-world environment is a nontrivial task independent of which currently available evolutionary techniques are employed. Evolving such behaviors using the simulation just described was even more difficult because, in order to be reliably fit, controllers had to evolve to cope with a whole variety of slightly different base set aspects of the simulation rather than just the one base set of robot-environment interactions present in the real-world situation. For this reason, although the evolutionary machinery used in Harvey et al. (1994) (control architectures, genetic algorithm, fitness function, etc.) was reimplemented initially for the experiments described here (in order to provide a direct comparison), it later was abandoned: Reliably fit individuals failed to evolve run after run, and the implication was that the control architectures used in the original experiments were just not capable of displaying the level of robustness necessary to cope with the uncertainty inherent in the simulation.

Figure 14 shows a typical example of the type of control architecture used in the experiments reported here. Functionally, they are very similar to those used in the Khepera robot experiments described in section 4: Weights on links are in the range of ± 2, and thresholds are in the range of 0 to 1. The activation function of every unit *including* the motor neurons was that of Equation 2. In addition to a genetically determined number (with a maximum of 3) of connections to each neuron from other neurons in the network, neurons also could receive normalized input, in the range of 0 to 1, from a camera image pixel. Motor signals were calculated from the output values of the four larger corner neurons of Figure 14, according to the relation $signal = 2 \times (A_1 - A_2)$, where A_1 and A_2 are the output values of the appropriate forward and backward neurons. The entire network, including inputs and outputs, and therefore the entire simulation, was updated at a speed of ten times per simulated second.

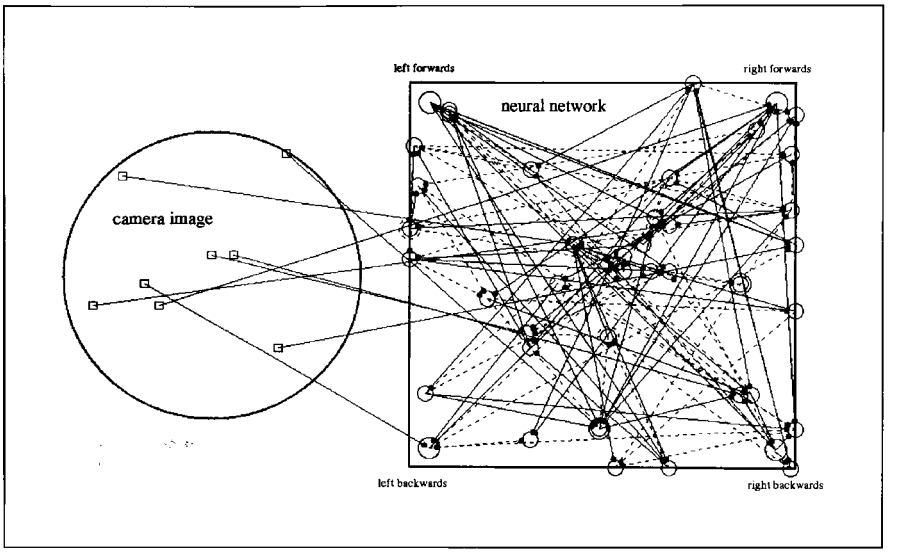


Figure 14
An example of a typical neural network evolved for the triangle-square discrimination task. On the left, the circular camera image, 8 pixels of which have been genetically specified as inputs to the neural network. On the right, the square box containing the 47 neurons of the network. Solid lines denote excitatory connections, and dashed lines denote inhibitory connections. The slightly larger units in each of the four corners are motor neurons.

The encoding scheme was chosen to allow genotypes to grow under genetic control with a minimum amount of phenotypic disruption, thus allowing arbitrary levels of complexity to evolve according to species adaptation genetic algorithm–like (SAGA-like) principles (Harvey, 1992).

Development took place in a two-dimensional space, the position of each neuron (apart from the four motor neurons; see Fig. 14) being genetically determined within that space. Each link within the network to any particular neuron was genetically specified in terms of the desired position of the neuron from which the link originated. The nearest neuron to this desired position, within a certain radius, then was allotted as the originator of the link. If no neurons lay within this radius, which was set at approximately an eighth of the width of the space, then the link failed to connect. In this way, the resultant network was independent of the exact order in which the connectivity of each neuron was worked out and developed. Because each neuron was encoded by a single gene on the genome, its connectivity was independent of the exact location of its gene on the genome and therefore was minimally disrupted by changes to this location due to the addition or deletion of extra genetic material.

Each gene was 15 integers long, each integer lying between 0 and 99. Apart from the first four genes, which specified the characteristics of the four positionally fixed motor neurons, the first two numbers of each gene specified the x and y coordinates of the corresponding neuron's position within the developmental space. The next number specified whether or not a neuron received input from a pixel of the camera image, with a probability of 1 in 4, and the next two numbers specified the x and y coordinates within the camera image of any pixel input. The sixth number of each gene specified the threshold, between 0 and 1, of the corresponding neuron. The last nine numbers specified the characteristics of up to three possible links *to* the relevant neuron *from* other neurons in the network: three numbers per link. The first two governed from which neuron the link originated by encoding the x and y coordinates of a point within the developmental space. The link then was judged to have originated from the nearest neuron to this point, or not at all if there were no neurons within a range of approximately one-tenth of the width of the space. The third of the three numbers specified the weight on the link between ± 2.

The genetic algorithm used in the experiments was extremely simple. After testing every member of a population of 100 individuals, the fittest 25 were used to produce the next generation by randomly picking parents and producing offspring until the new population was full. Crossover was applied with a frequency of 0.7. Each mutation involved altering one of the integers that made up the genome by an amount taken from a normal distribution with a mean of 0 and a standard deviation of approximately 10. Mutated values then were clipped to lie within the range of 0 to 99. The expected number of mutations per genotype, according to a Poisson distribution, was 1. At each offspring event, probability that a random gene would be introduced into the offspring genotype was 0.02, as was the probability that an already existing gene would be deleted.

The fitness function returned the average value scored by an individual in a total of eight fitness trials, each trial lasting a maximum of 20 simulated seconds. For the first set of four trials, the triangle was on the left and the square was on the right and, for the second set of four trials, the triangle was on the right and the square was on the left. For both sets, the robot was started at each one of the four starting positions shown in Figure 11 in turn. At the end of each trial, when either the time had run out or the robot had hit a wall, the fitness function returned $100 - d$ as the fitness score, where d was the distance from the center of the robot to the center of the triangle.

5.4 Experimental results

Figure 14 shows a typical example of the sort of network that evolves to be reliably fit within the simulation. This particular network is the result of nearly 6000 generations

of the genetic algorithm (approximately 12 hours as a single user on a SPARC Ultra), which is the simulated equivalent of $6000 \times 100 \times 8 \times 20 = 96{,}000{,}000$ seconds, or more than 3 years worth of real-world evolution. When placed in one of the four starting positions in the arena, the network initially causes the robot to turn in a tight circle clockwise. If the square comes into the view of the camera, the rotational speed of the robot actually increases until the square is out of view. When the triangle hoves into view, the robot "locks on" and proceeds directly toward it, adjusting its course as it goes.

To determine whether it would cross the reality gap, the network was downloaded onto the gantry and tested continuously[7] and automatically on the triangle-square task in the real world under full disco lighting. In total, 200 trials were performed: 100 for the triangle on the left and the square on the right, and 100 for the triangle on the right and the square on the left. At the beginning of each trial, the robot was started in one of four different starting positions, corresponding to those of the simulation, and these were run through in cycle from trial to trial. On each trial, the robot was automatically judged to have achieved the task successfully if, by the end of the trial, it was stationed within a rectangular area extending approximately 10 cm to either side of the triangle and 15 cm out into the arena (see Fig. 11). Inspection revealed that this automatable criterion corresponded well with more subjective notions of success and failure on the task.

With the triangle on the right and the square on the left, the robot performed the task successfully 98 times out of 100. Of the two failures, one occurred when the gantry rails were being polished (to try to prevent the motors from jamming) and the lights were temporarily obscured by the author's body. The other failure is difficult to explain, as the gantry robot just headed off into a wall under otherwise unremarkable circumstances. This may have been due to freak noise but may also have been due to a mechanical or software error.

With the triangle on the left and the square on the right, the robot performed the task successfully 97 times out of 100. All three failures occurred from the same starting position furthest from the triangle and, in each case, the circumstances were similar. Having turned away from the wall, the robot failed to lock on to the triangle but continued spinning on the spot. It would spin past the square, past its original starting orientation, and back around to face the triangle. In two of three of the cases, it then locked on to the triangle and started to move directly toward it, running out of time before it reached the success zone. In the third case, it failed to lock on again and ran out of time before it could spin right around to face the triangle for

7 In practice, because of the propensity of the mechanics of the gantry robot to cease and the software controlling it to crash, the testing procedure had to be watched continuously and restarted (from where it had crashed) on a number of occasions.

a third attempt. In all three cases, if more time had been allowed, the robot would almost certainly have reached the target.

6 Conclusions

In the first part of this article, a theoretical investigation was made into the circumstances under which evolved robot controllers are able to cross the reality gap. It was suggested that a sufficient condition is that evolved controllers are both base set exclusive and base set robust. The radical envelope-of-noise hypothesis then was stated: If random variation is applied in specific ways to all aspects of the simulation, then control architectures that evolve to be reliably fit within the simulation will be base set robust and base set exclusive and therefore will successfully cross the reality gap. It was further argued that if the hypothesis were well founded, then it would be possible to create minimal simulations that were easy to build and computationally cheap.

The second part of the article detailed two sets of experiments that together provide some evidence for the hypothesis. In the first set, controllers with internal state were evolved to solve a simple T-maze task using a minimal simulation of a Khepera robot. In the second set, controllers were evolved to discriminate visually between two shapes using the Sussex University gantry robot. In both cases, evolved controllers were able successfully to cross the reality gap, exhibiting extremely robust behaviors when downloaded onto the real robots. In particular, the controllers evolved for the gantry robot performed significantly better than any others that had been evolved previously using alternative evolutionary methodologies.

In Matarić and Cliff (1996), the authors suggest that as robots and the behaviors we want to evolve for them become more and more complex, simulations will become either so computationally expensive that all speed advantages over real-world evolution will be lost or so difficult to design that the time taken in development will outweigh the time saved in reality. This article has contributed to showing that for certain types of behaviors and robots, at least, this will *not* be the case. The reasons for this are explained briefly here.

First, for complex behaviors: The experiment with the Khepera robot (section 4) does not involve the evolution of a behavior that is particularly complicated in itself, but it does prove that it is possible to evolve complicated behaviors in a minimal simulation (at least as far as the simulation is concerned). To illustrate this, consider a slightly extended version of the minimal simulation described in section 4. In this version, the Khepera robot is not presented with merely a single junction at the end of the first corridor but with a whole series of turnings and junctions that together add up to a complex maze. In the first corridor, furthermore, the Khepera does not

pass just a single light signal but a whole series, placed one after another, some on the left and some on the right, that together signal the correct path through the maze that follows. This behavior would be extremely complicated by today's standards of what can and cannot be evolved, and yet the minimal simulation would remain simple and fast. It therefore is possible to build minimal simulations for the evolution of complex behaviors.

Second, for complex robots, we need only look at the experiments of section 5 for proof that it is possible to create minimal simulations for robots that employ complex sensory modalities such as vision. As for complex motor modalities, recent experiments carried out at Sussex involving a minimal simulation of an octapod robot have succeeded in evolving neural network controllers that allow the octapod not only to walk smoothly and robustly in a straight line but also to turn away from obstacles that it senses using its infrared sensors and to back away from obstructions that are sensed by way of its front bumper or whiskers (Jakobi, 1997b). It therefore is possible to create minimal simulations for robots that employ complex motor modalities.

The point is that whether a minimal simulation is easy to construct and runs fast depends not on the complexity of the behavior we want to evolve when using it, nor on the complexity of the robot that it simulates, but only on the complexity of the base set of robot-environment interactions necessary to underlie the behavior. Provided these are simple enough, then the behavior or the robot (or both) can be arbitrarily complex.

Worries as to whether minimal simulation techniques will scale up can therefore be reduced to worries about whether the robot-environment interactions employed by the robots and control architectures of the future will be prohibitively complex. It is too early to say whether this will or will not be the case, but consider two points: (1) that results in insect and invertebrate neuroscience suggest that many complex behaviors often are accomplished by way of simple interactions with the environment rather than complicated ones (Wehner, 1987; Horridge, 1992; Collett, 1996); and (2) that control strategies grounded in complex robot-environment interactions can lead to prohibitively heavy real-time processing requirements (Brooks, 1991). This latter fact has fueled the trend in mobile robotics over the last few years from the internal world model–making robots of the seventies (Nilsson, 1984) to the current low-level behavior-based robotics of the present day (Chiel, Beer, Quinn, & Espenschied, 1992).

Acknowledgments

I would like to thank all the members of the Evolutionary and Adaptive Systems group at COGS for various crucial discussions. Special thanks go to Phil Husbands

for patient proofreading and after-hours consultation, to Peter Stuer for arguing that it wasn't possible, and to Tim Smithers for seeing that it might be. Thanks also to my two anonymous reviewers for their helpful comments and to the School of Cognitive and Computing Sciences (COGS) itself for the bursary that allows me to undertake this work.

References

Beer, R., & Gallagher, J. (1992). Evolving dynamic neural networks for adaptive behavior. *Adaptive Behavior, 1*, 91–122.

Braitenberg, V. (1984). *Vehicles: Experiments in synthetic psychology*. Cambridge, MA: MIT Press/Bradford Books.

Brogan, W. L. (1991). *Modern control theory* (3rd ed.). New York: Prentice Hall.

Brooks, R. (1991). Intelligence without reason. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*. San Mateo, CA: Morgan Kauffman.

Chiel, H., Beer, R., Quinn, R., & Espenschied, K. (1992). Robustness of a distributed neural network controller for locomotion in a hexapod robot. *IEEE Transactions on Robotics and Automation, 8*(3), 293–303.

Collett, T. S. (1996). Insect navigation en-route to the goal: Multiple strategies for the use of landmarks. *Journal of Experimental Biology, 199*(1), 227–235.

Collins, R., & Jefferson, D. (1991). Selection in massively parallel genetic algorithms. In R. K. Belew & L. B. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms, ICGA-91*. San Mateo, CA: Morgan Kaufmann.

Floreano, D., & Mondada, F. (1994). Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In D. Cliff, P. Husbands, J. Meyer, & S. Wilson (Eds.), *From animals to animats 3: Proceedings of the Third International Conference on the Simulation of Adaptive Behavior*, Vol. 3. Cambridge, MA: MIT Press/Bradford Books.

Harvey, I. (1992). Species adaptation genetic algorithms: The basis for a continuing saga. In F. J. Varela & P. Bourgine (Eds.), *Toward a practice of autonomous systems: Proceedings of the First European Conference on Artificial Life*. Cambridge, MA: MIT Press/Bradford Books.

Harvey, I., Husbands, P., & Cliff, D. (1994). Seeing the light: Artificial evolution, real vision. In D. Cliff, P. Husbands, J. Meyer, & S. Wilson (Eds.), *From animals to animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, Vol. 3. Cambridge, MA: MIT Press/Bradford Books.

Horridge, G. (1992). What can engineers learn from insect vision. *Philosophical Transactions of the Royal Society of London, 337*(1281), 271–282.

Husbands, P., & Harvey, I. (1992). Evolution versus design: Controlling autonomous robots. In *Integrating perception, planning and action: Proceedings of the Third Annual Conference on Artificial Intelligence, Simulation and Planning*. Los Alamitos, CA: IEEE Press.

Husbands, P., Harvey, I., & Cliff, D. (1993). An evolutionary approach to situated a.i. In A. Sloman, D. Hogg, G. Humphreys, A. Ramsay, & D. Partridge (Eds.), *Prospects for artificial intelligence*. Amsterdam: IOS Press.

Husbands, P., Harvey, I., Jakobi, N., Thompson, A., & Cliff, D. (1997). A case study in evolutionary robotics. In T. Back, D. Fogel, & Z. Michalewicz (Eds.), *Handbook of evolutionary computation*. New York: Oxford University.

Jakobi, N. (1996). Encoding scheme issues for open-ended artificial evolution. In H.-M. Voigt, W. Ebeling, I. Rechenberg, & H.-P. Schwefel (Eds.), *Proceedings of the Fourth International Conference on Parallel Problem Solving in Nature*. Berlin: Springer-Verlag.

Jakobi, N. (1997a). Half-baked, ad-hoc and noisy: Minimal simulations for evolutionary robotics. In P. Husbands & I. Harvey (Eds.), *Proceedings of the Fourth European Conference on Artificial Life*. Cambridge, MA: MIT Press.

Jakobi, N. (1997b). A minimal simulation for evolving walking behaviour in an octopod robot. Cognitive science research paper CSRP464, University of Sussex.

Jakobi, N., Husbands, P., & Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. In F. Moran, A. Moreno, J. Merelo, & P. Chacon (Eds.), *Advances in artificial life: Proceedings of the Third European Conference on Artificial Life*. Berlin: Springer-Verlag.

K-Team (1993). *Khepera user's manual*. Lausanne: EPFL.

Matarić, M., & Cliff, D. (1996). Challenges in evolving controllers for physical robots. *Robot and Autonomous Systems, 19*(1), 67–83.

Michel, O. (1995). An artificial life approach for the synthesis of autonomous agents. In J. Alliot, E. Lutton, E. Ronald, M. Schoenauer, & D. Snyers (Eds.), *Proceedings of the European Conference on Artificial Evolution*. Berlin: Springer-Verlag.

Miglino, O., Lund, H., & Nolfi, S. (1995). Evolving mobile robots in simulated and real environments. *Artificial Life, 2*(4), 417–434.

Nilsson, N. J. (1984). Shakey the robot (Tech. Note 323). Menlo Park, CA: SRI International.

Nolfi, S., Floreano, D., Miglino, O., & Mondada, F. (1994a). How to evolve autonomous robots: Different approaches in evolutionary robotics (Tech. Rep. PCIA-94-03). Rome, Italy: Department of Cognitive Processes and Artificial Intelligence.

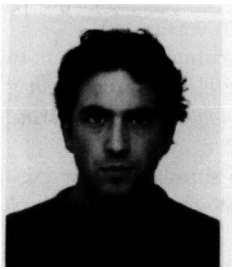
Nolfi, S., Miglino, O., & Parisi, D. (1994b). Phenotypic plasticity in evolving neural networks. In P. Gaussier & J.-D. Nicoud (Eds.), *Proceedings of the "From Perception to Action" Conference*. Cambridge, MA: IEEE Computer Society Press.

Thompson, A. (1995). Evolving electronic robot controllers that exploit hardware resources. In F. Moran, A. Moreno, J. Merelo, & P. Chacon (Eds.), *Advances in artificial life: Proceedings of the Third European Conference on Artificial Life*. (Lecture Notes in Artificial Intelligence 929.) Berlin: Springer-Verlag.

Thompson, A. (1996). An evolved circuit, intrinsic in silicon, entwined with physics. In T. Higuchi & M. Iwata (Eds.), *Proceedings of the First International Conference on Evolvable Systems (ICES'96)*, LNCS. Berlin: Springer-Verlag.

Wehner, R. (1987). Matched-filters—neural models of the external world. *Journal of Comparative Physiology, 161*(4), 551–531.

About the Author



Nick Jakobi

Nick Jakobi has received an MA in philosophy and mathematics from the University of Cambridge and an MS degree in knowledge-based systems from the University of Sussex. The research toward his doctoral degree at the University of Sussex, which he is currently completing, has centered around the theory and practice of artificially evolving control architectures for real robots.