Evolutionary Robotics and the Radical Envelope of Noise Hypothesis

Nick Jakobi
School of Cognitive and Computing Sciences
University of Sussex, Brighton BN1 9QH, England
email: nickja@cogs.susx.ac.uk
tel: (UK) 01273 678061
fax: (UK) 01273 671320

March 12, 1997

Abstract

For several years now, various researchers have endeavoured 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 paper outlines new ways of thinking about and building simulations upon which such assessments can be performed. It puts forwards 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. As long as simulations are built according to this methodology, it is hypothesised, then it does not matter how inaccurate or incomplete they are: controllers that have evolved to be reliably fit in simulation will still 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, and in the second set, controllers were evolved for the Sussex University gantry robot that were able to visually distinguish a triangle from a square, under extremely noisy real world conditions, and steer the robot towards it. In both cases, controllers that were reliably fit in simulation displayed extremely robust behaviour when downloaded into reality.

1 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 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 be done in only one of two ways: control architectures must either 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 have their problems.

As Mataric and Cliff (1996) point out, the evaluation of control architectures on real robots must be done in real time, and this makes the entire evolutionary process prohibitively slow. As an example, they cite the evolution of collision-free navigation on a Khepera robot, which in the experiments reported in Floreano and Mondada (1994) took a total of 65 hours (100 generations at 39 minutes a generation) to evolve[1]; it is hard to see how this approach can scale up, if the behaviours we are after require thousands or even millions of generations. But even if we are resigned to an evolutionary process that takes years rather than days, then there are different problems that must be faced. The process must be automated, which begs questions about how data is to be collected for fitness evaluations and the robot placed at the start of fitness trials and so on without human intervention. Power must be supplied continuously to robots in situations where batteries have limited life-spans and tethering by a permanent power lead is not always possible. And 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, since 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), Miglino, Lund, and Nolfi (1995) have shown that it is possible to evolve control architectures in simulation for a real robot. Now this is no longer in doubt the question becomes one of whether the technique will scale up. In Mataric and Cliff (1996), the authors argue that if behavioural transference can only be guaranteed when a carefully constructed empirically validated simulation is used, then as robots and the behaviours 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.
- so hard 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 upon and using which fast-running simulators can be easily and cheaply built that guarantee the transference of evolved behaviours from simulation to reality.

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

This paper puts forwards such a theoretical and methodological basis, albeit at a preliminary stage, and outlines some notable experimental successes using the techniques proposed. Section 2 undertakes a conceptual analysis of how it is possible for control architectures that have evolved in simulation to transfer into reality in the first place. Two conditions are put forwards that must be true of evolved controllers if this transference is to be successful. Section 3 outlines a methodology for building simulations within which reliably fit control architectures are guaranteed to meet these two conditions, and are therefore guaranteed to cross the reality gap. The idea of fast, easy to build minimal simulations is also 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.

2 How is crossing the reality gap possible in the first place?

As has been demonstrated in several papers (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 papers as to why behaviours successfully transfer to reality when evolved under certain simulation conditions while not under others fall well short of the level of understanding necessary for the development of a general simulation building methodology. The consensus view seems to be that control architectures will successfully transfer if the right amount of noise is included in a carefully constructed and empirically validated simulation of the robot and its environment[2]. But there is no such thing as the perfect simulation; some real-world features will be modelled at the expense of others. And since *my* 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 counts as success?

If we are to come up with a general methodology for building simulations for evolutionary robotics it is important to define exactly what we mean when we say that a behaviour 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 and 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 qualitatively similar in reality to how they behave in simulation, but again, as we shall see, this is not always possible either. For the purposes of this paper, a control architecture is said to have successfully crossed the reality gap if it successfully displays the behaviour it was evolved to display when down-loaded

[2] Although the nature of the 'right amount of noise', and indeed even what it means for a behaviour to 'successfully transfer from simulation to reality', varies markedly between papers on the topic

into reality. As in Jakobi et al. (1995) this may often be a somewhat subjective measure, but should nevertheless be uncontroversial. If I use a simulation to evolve robot controllers to move round a cluttered environment avoiding objects, for example, then control architectures successfully cross the reality gap if, when downloaded, they do indeed cause a robot to move round a cluttered environment avoiding objects. If I use a simulation to evolve visually guided robot controllers that steer towards a target, then control architectures successfully cross the reality gap if, when downloaded, they do indeed steer the robot towards 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 entire universe. If it was, 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 entire universe on two counts: it will only model a finite set of real-world features and processes, and those that it does model, it will model inaccurately. Both of these failings have fundamental implications for how 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 them both in turn.

Simulations can't accurately model everything

Since a simulation can only model a finite set of real-world features and processes, the first thing that we must decide upon when building one is exactly what this set should consist of. Obviously, for the purposes of evolutionary robotics, we are only interested in those real-world features and processes that have some bearing on a particular robot or robots and a particular environment. The question still arises, however, as to which of these we must model, and which we can get away with leaving out, in order that evolved controllers will cross the reality gap. There is little point in modelling the colour of a robot or the objects in its environment, for example, if that robot is only fitted with sonar range sensors.

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

The conventional approach is to try and model, as accurately as possible, *all* of the real-world features and processes that could conceivably affect a robot's behaviour. The rationale behind this being that again (ignoring the inevitable inaccuracy of the

modelling 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 evolving controllers come to depend upon, they will also 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 only hope to capture a subset of the totality of possible robot-environment interactions. They should therefore be thought of in the same way as the simulations discussed above.

Because these concepts are central to the ideas put forwards below, we will give them a special terminology. Throughout the rest of the paper, the set of robot-environment interactions that we decide to model as a basis for the behaviours 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 infra red 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 environment. Those aspects of the simulation that correspond to members of this base set, and that evolving controllers can safely depend upon and use without encountering any differences between simulation and reality, will be referred to as the *base set aspects* of the simulation. It is also likely that there will be aspects of the simulation which behaviours of evolving controllers can depend upon 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 accurately modelled in the simulation, then those that are not modelled will often leave 'gaps' that must be filled in by arbitrary values and processes in order that the simulation constitutes a complete virtual reality. Those aspects of the simulation that are not base set aspects, but that evolving control architectures can come to depend upon nevertheless, will be referred to throughout the rest of the paper as the *implementation aspects* of the simulation. We will examine concrete examples of all of these terms in Section 3.

To see why this is important, consider a hypothetical simulation in which we have decided upon a particular base set of robot-environment interactions and modelled them 100% accurately. In other words, we have created a simulation whose base set aspects are 100% 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 behaviour, *and those base set aspects alone* (either implicitly or explicitly), then it will successfully cross the reality gap. This is for the simple reason 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 universe.

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

Simulations can't accurately model anything

Even if we know the conditions under which robot controllers will cross from a simulation whose base set aspects are 100% accurate into reality, we are only half way towards a general explanation. It just is not possible to model a base set of robot-environment interactions with 100% accuracy. To explain how control architectures can and do cross the reality gap, we need something extra to the exclusivity condition laid out above. 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 above, for a control architecture to cross the reality gap does not mean that it performs the task identically in the real world to the way that it does in simulation but only that it does indeed perform the task in the real world. Thus small inaccuracies in the base set features of a simulation may result in slight differences between a controller's behaviour in simulation and its behaviour 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 shot-gun at five paces. If your aim is off by a metre or so, you'll still hit it and this is all we are after. Of course in more complicated and involved situations there will not be so much room to manoeuvre, but it should be kept in mind that even for the most delicate of behaviours there will normally be a little bit of slop that can soak up small discrepancies.

Some control architectures, however, will be more robust to inaccuracy than 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 which employ feedback or similar techniques will be far more robust to inaccuracy and noise than those that don't (Brogan, 1991), and this is true here also; non-brittle control strategies and behaviours 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 above.

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 behaviour, 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 come up with), in which case there is no need to focus too hard on exact mechanisms by which control architectures may be robust, since 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 behaviours if they are to perform satisfactorily in reality. Throughout the rest of the paper, we will refer to controllers that display this property as being *base set robust*.

3 A new methodology for building simulations

In the previous section, two properties were put forward that together are sufficient for the successful transfer of robot controllers across the reality gap: they must be base set exclusive, and they must be base set robust. In this section ways of forcing the evolution of these two properties by placing certain restrictions on the simulation itself are put forward. First of all we will examine ways of ensuring that successful evolved behaviours are base set exclusive, and then we will examine ways of ensuring that successful evolved behaviours are base set robust.

3.1 Ensuring that reliably fit controllers are base set exclusive

When evolving control architectures 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 to consistently and robustly 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 behaviour 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. In order to ensure that successful evolved behaviours are base set exclusive, therefore, we need to ensure that reliably fit individuals do not rely in any way on implementation aspects of the simulation to achieve reliable high fitness, but only on base set aspects.

This can be done by making all the implementation aspects of a simulation *unreliable* by varying them randomly from trial to trial. In such a situation, the only 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 ie. the base set aspects. If the behaviour of a particular control architecture depends on an unreliable aspect of the simulation then it too will be unreliable. Since 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 behaviour, and will therefore be base set exclusive.

The hardest 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 modelling process, in which case they can be quite subtle and hard to spot. For example, one of the robot-environment interactions we might chose 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 chose 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 can

not 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 since we have put them in especially to make the modelling 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 very hard in others. In order 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 must 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 over-zealous in our lust for computational expediency, that we may effectively exclude so many real-world features from the simulation that what we are left with is insufficient for successful behaviour.

Once 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 to just add large amounts of noise to everything which 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 randomly vary the implementation aspects of the simulation from trial to trial as opposed to just during each trial. Since 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. In some ways this is a problem of generalisation, but the point is not to subject evolving controllers to every possible case of every implementation aspect, so that controllers evolve to deal with each one in turn. It is sufficient to make these aspects *unreliable enough* in order that successful controllers ignore them altogether, employing strategies which are independent of them, and achieving high fitness by exclusively using the base set aspects of the simulation.

3.2 Ensuring that reliably fit controllers are base set robust

In order to ensure that reliably fit control architectures are base set robust, we must be able to ensure that they are able to cope with the differences between the base set of robot-environment interactions upon which a particular simulation is founded and

the base set aspects of that simulation. We may do this by adapting ideas borrowed from Husbands and Harvey (1992) (with further elaborations in Husbands, Harvey, and Cliff (1993)). The idea 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 favour of controllers that are better able to cope with slightly different versions of each base set aspect, and thus in favour 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. So in order to *ensure* that reliably fit individuals will be base set robust, all we need is some way of knowing how much random variation it is necessary to apply to the base set aspects of the simulation, and the best way in which to apply it.

As has already been said, it is rarely possible to simulate even the smallest portion of the world completely accurately. However, it is also rare that the simulation builder will not have at least some idea of how *inaccurate* their 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 learnt from experiments reported in Jakobi et al. (1995). In these 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 down-loaded 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 to the real base set of robot-environment interactions ie. they were much more noisy. Given this fact, it is unsurprising that evolved controllers were unable to cross the reality gap since 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.

In order for their to be a selection pressure in favour of controllers that can cope with slightly differing versions of each base set aspect of a simulation, they need to be varied *between* trials, and not during them. There should of course be noise on sensors and actuators during each trial, since 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 upon 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 be left steady throughout the evolutionary process at unrealistic levels.

3.3 The radical envelope of noise hypothesis

The suggestions outlined above together add up to what we shall call the radical envelope of noise hypothesis. As opposed to a single bold statement of fact which may be proved true or false by empirical testing, it is rather a proposed methodology using which we can build simulations that are posited to display certain properties. Whether or not the hypothesis is true (or, more pragmatically, useful), depends on whether reli-

ably fit control architectures, that have evolved in simulations built using the methods and techniques outlined above, transfer across the reality gap.

To sum up the methodology:

- A base set of robot-environment interactions (that are sufficient to underly the behaviour we want to evolve) must be identified and made explicit. A basic simulation should then be constructed that includes a model of these interactions. This simulation will display base set aspects, that have a basis in reality, and implementation aspects, that do not.

- Every implementation aspect of the simulation must be randomly varied from trial to trial so that reliably fit controllers are base set exclusive. In particular, *enough* variation must be included so that the only practicable evolutionary strategy is to actively ignore each implementation aspect entirely.

- Every base set aspect of the simulation must be randomly varied from trial to trial so that reliably fit controllers are 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 robot-environment interaction model and reality, but not so large that reliably fit controllers fail to evolve at all.

Of course these guide-lines are very general, and not a step by step recipe using which one can be guaranteed success every time. It is hoped, however, that together with the example simulations of section 4 and section 5 the reader is left with at least some idea of how they might go about constructing good simulations for evolutionary robotics.

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 bear minimum. This is where the potential power of the radical envelope of noise hypothesis lies. 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 above. What *is* much more unlikely in this situation, is that reliably fit controllers will evolve at all. There will always be limits to the amount of randomness that the evolutionary machinery can find ways of coping with, 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 will just fail to evolve.

However, if the evolutionary machinery *is* sufficiently powerful we can evolve complex control architectures, capable of performing non-trivial 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

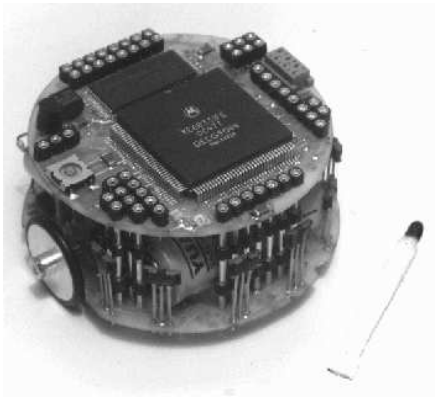

Figure 1: The Khepera robot.

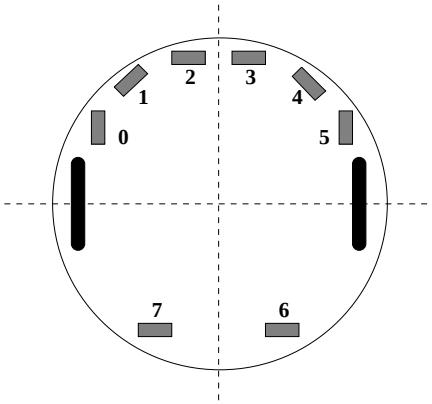

Figure 2: A plan of the Khepera robot showing the positions and numbers of the infra red sensors and the two wheels.

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

Both of the experiments described below involve minimal simulations of this type. 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.

4 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 5.8 cm in diameter and about 3 cm high. It has eight infra red sensors, which respond to nearby objects, 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; Miglino et al., 1995; Michel, 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 behaviour for the Khepera robot that was at least one step up from the simple reactive behaviours that have been prevalent

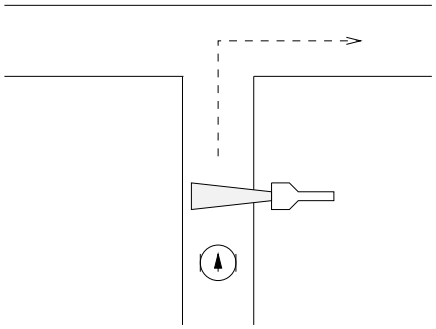

Figure 3: The task in the real world.

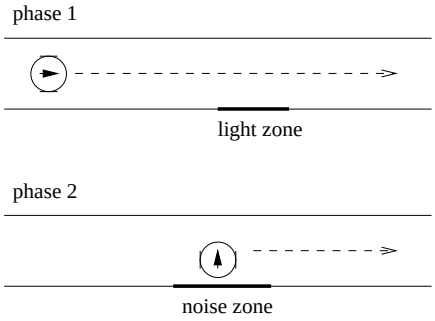

Figure 4: The task in simulation.

in the Evolutionary Robotics literature so far. The behaviour that was decided on 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, turn down the corresponding arm of the T-maze. This behaviour 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 behaviours both), but they must also involve some internal state that allows them to 'remember' which side the lamp was on so that they can take the correct turning at the junction.

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 about 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 behaviours. 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 infra-red 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 number 2 of the above list seems totally counterintuitive since a T-maze has nothing to do with infinite corridors. However, with respect to the infra-red sensors of a Khepera which have a maximum range of only about 8cm, 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 randomly varied from trial to trial according to the methodology laid out above. In this way, reliably fit controllers were forced to use strategies that depended on the interactions between the infra-red 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.

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 travelled a predetermined distance, it was suddenly popped out of the first corridor, rotated through ninety degrees, and popped into the middle of a second corridor for phase two. It then had to chose 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, it has enough in common that evolving control architectures which are prohibited from relying on any of the differences are still 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, travelling down a straight corridor, is confronted with a wall straight across its path and a second corridor stretching off to either side were all modelled.

The differences between the simulation and the real world T-maze all occur around the T-junction. When the virtual Khepera suddenly appears in the second corridor facing the wall at the start of phase 2, there is a continuous wall directly behind it (see figure 4). In reality, when the Khepera is confronted with a wall across its path and forced to make its decision on which way to turn, there is a complicated junction in the wall behind it (see figure 3). Because of this, the simulated robot's infra red sensor interactions with the simulated back wall, in an area corresponding to where the corridors meet in reality and about 5cm to either side, were regarded as implementation aspects. If this section of the wall fell within range of the infra-red sensors then how 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.

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

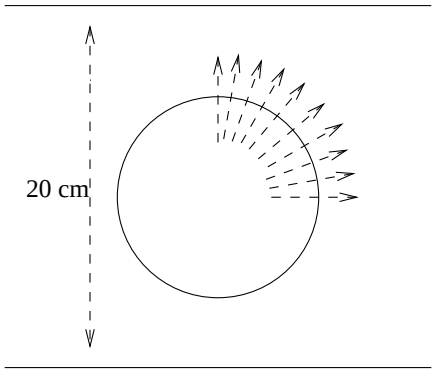

Figure 5: A look up table contains the perpendicular distances to the walls of a 20cm wide corridor for all eight sensors in ten possible orientations.

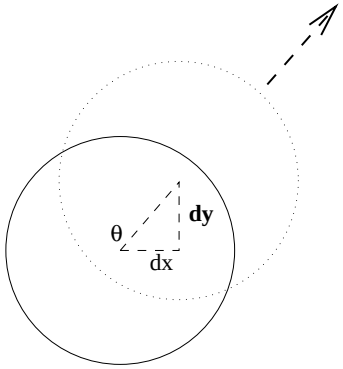

Figure 6: A look up table holds horizontal and vertical increment values for 36 different orientation values and an average speed of 1.

implementation aspects concerned with the differences between the simulation and reality around the area of the T-junction, there were also several others:

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

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

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 infra-red sensors. The third robot-environment interaction listed above, namely the way in which the ambient light sensors react to bright verse ambient light levels, was actually 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 a second. Figure 6 shows how the new position of the virtual Khepera 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 travelling at a speed of 1cm per second. To work out its new position, the values returned from this look up table were multiplied by the average wheel speed in cm per second. The speed of each wheel was calculated directly from multiplying the 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 (about 5.2cm). There was no allowance for momentum and the noise inherent in the real-world situation was not modelled.

Calculating the infra-red values was a slightly more involved process and proceeded in three stages. Firstly, the robot's orientation was used to generate rough distance to wall metrics for each sensor, as if the robot was in the centre of a 20cm wide infinite corridor. Secondly, these vales were scaled according to the actual width of the corridor in the simulation, and the distance from the robot to each wall. And thirdly, the scaled distance to wall metrics were used to calculate infra red sensor values by way of a simple linear relationship. This process is described in more detail below.

Figure 5 demonstrates what the values held in the infra-red look-up-table represented. There were 10 sets of 8 values, each set corresponding to one of 10 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 centre of the robot (which is 10cm away from each wall), along the lines of the corresponding sensors, to the walls of an infinitely long corridor. If the distance from the centre of the robot, along the line of a sensor, to a wall of the corridor is d, then the warped distance value wdv stored in the look-up-table was given by:

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

This warped distance value, which was always shorter than the line of sight distance, accounted in a very approximate way for the fact that the infra-red sensors of a Khepera are sensitive over a whole arc rather than just along the direct line of sight extending out from each sensor (K-Team, 1993).

The minimum possible value stored in the table, therefore, for a sensor directly facing the wall, was 10cm. The maximum possible value, for a sensor directly facing down the corridor, was infinity. Now although there were only 10 sets of values stored in the table (one set of 8 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. This was done 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 was then used to ascertain a set of values and these were reflected across the mid-line of the robot if necessary (ie. if the angle was between 90 and 180 or between 270 and 360). If the robot was in the centre of a 20cm wide, infinitely long corridor, therefore, warped distance values could be calculated for any sensor at any orientation.

In practice, the perpendicular distance from the centre 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

10cm. For example, if the distance from the robot to a wall was actually 5cm instead of 10cm, 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 above, from the centre 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 was less than b (typically about 5), would return zero if its warped distance value was greater than a (typically about 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 infra-red 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 any other aspect.

The way in which ambient light sensors respond to bright versus ambient light levels was modelled 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 around 450 to a value of around 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 which of the two directions directly down the corridor the robot was closest to pointing. Random deviates in the range ± 50 were added to each ambient light sensor value at each iteration.

Ensuring that reliably fit behaviours 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 was done in two ways in the simulation under discussion. Firstly, random offsets of between ± 1cm per second were generated at the beginning of each trial, and added to wheel-speeds during position update calculations. This had the effect of causing 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 come up with ways of coping with and being robust to a variety of different movement characteristics. Secondly, the constants a and b of equation 4.2 were randomly set 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 come up with strategies that were robust to a range of infra-red sensor characteristics.

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's all important infra red sensors) almost everywhere. With this in mind, a simulation of a T-maze was constructed whose base set aspects could be generated using a simple model of a Khepera's interactions with the walls of a continuous infinite corridor. The nature of these modelled interactions was varied slightly and randomly, from trial to trial, in order to ensure that reliably fit controllers were base set robust. Those aspects of the simulation that corresponded to sections of the T-maze which could not be modelled 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 in order 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 ie. whether control architectures that have evolved to reliably perform a task in a simulation created according to the methodology outlined in section 3 would successfully transfer to reality. However, for the T-maze task described above, it is no simple matter to evolve reliably fit controllers in the first place, and careful attention had to be paid to the choice of evolutionary machinery. This machinery is briefly described here.

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[3]. All non-motor neurons had simple step threshold activation functions of the form:

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

where A_j was the activation of the j-th neuron, T_j was the threshold of the j-th neuron, and w_{ij} was the weight on the connection from the i-th neuron to the j-th neuron. 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 ± 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

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

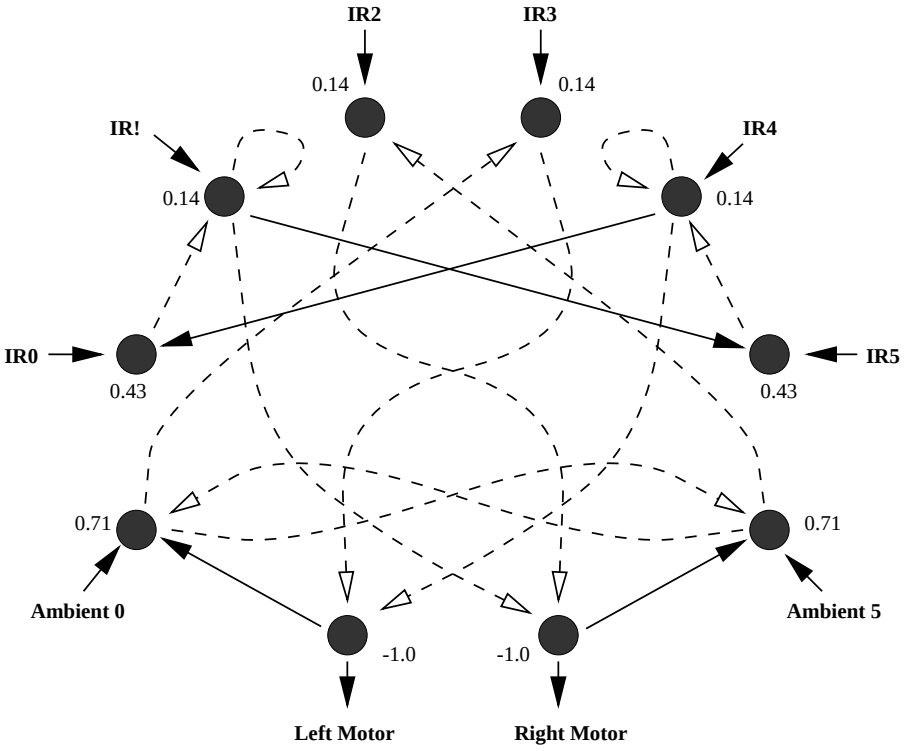

Figure 7: A typical evolved network. The solid arrows are excitatory links and the dashed arrows are inhibitory links; exact weight values are not shown. Threshold values appear next to each neuron.

output. All sensor values were normalised in the range 0 to 1 and motor outputs were multiplied by a factor of 10 to give motor signals in the range $\pm$8cm per second. The network, sensor values and motor outputs (in fact the entire simulation) were updated the equivalent of 10 times a 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 5 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 4 bits of each gene, a binary number between 0 and 16, defined the threshold of that neuron by normalising between $\pm$1. The next 3 sets of 8 bits defined the three possible links to that neuron from other neurons in the network: the first 4 ascribing one of 16 possible values for the weight of the link between $\pm$2 and the next 4 bits defining which of 16 neurons the link was from. Because there were only 10 neurons in total in the network, if a link indexed a non-existent neuron, then it did not connect, thus placing the number of links to a neuron under genetic control.

The fitness function returned the average value scored by an individual in a total

of ten fitness trials, each lasting the equivalent of fifteen seconds. At the end of each trial, the fitness value was equal to the total distance travelled 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 travelled 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 by 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 by 3 square centred on that location. Two probabilistically fit parents were chosen from this breeding pool according to a linear rank-based selection procedure, and an offspring 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 mutations per genotype, according to a Poisson distribution, was 2. At each offspring event, not only was the offspring's fitness evaluated, but both parents were reevaluated as well.

4.4 Experimental results

Figure 7 shows a typical example of the sort of neural network that consistently evolved within around 1000 generations (where a generation was taken to be 100 offspring events). This is the simulated equivalent of $300 \times 15 \times 10 \times 100 = 45000000$ seconds or over 17 months of continuous real-world evolution, and takes around 4 hours to run as a single user on a SPARC Ultra. The network reliably achieved near-optimal fitness within the simulation. In order to see whether it would successfully transfer across the reality gap, the network was downloaded onto a Khepera robot and tested as to its ability to perform the task in the real world. Sixty different trials were performed one after another, twenty in each of three different widths of corridors, with the light on the left for ten trials and the light on the right for the other ten. The consequent robot behaviours were filmed from above so that the exact path taken by the Khepera on each trial could be extracted using basic image-processing techniques and overlayed upon aerial views of the set-up. The results of this process are the six images of Figure 8.

In the top pair of images, the corridor is only 11cm wide and the paths taken by the Khepera on all twenty occasions are tightly constrained. In the second pair of images, where the corridor is 18cm wide, and especially in the bottom pair of images, where the corridor is 23cm wide, the paths taken by the Khepera are less constrained. The Khepera still turns the correct way at the T-junction, however, even though on several occasions it must turn through greater than ninety degrees in order to do so. Note that the path taken in most cases was near-optimal, and that in every case the task was performed satisfactorily: the criterion put forwards in section 2 for a control architecture to successfully transfer from simulation to reality.

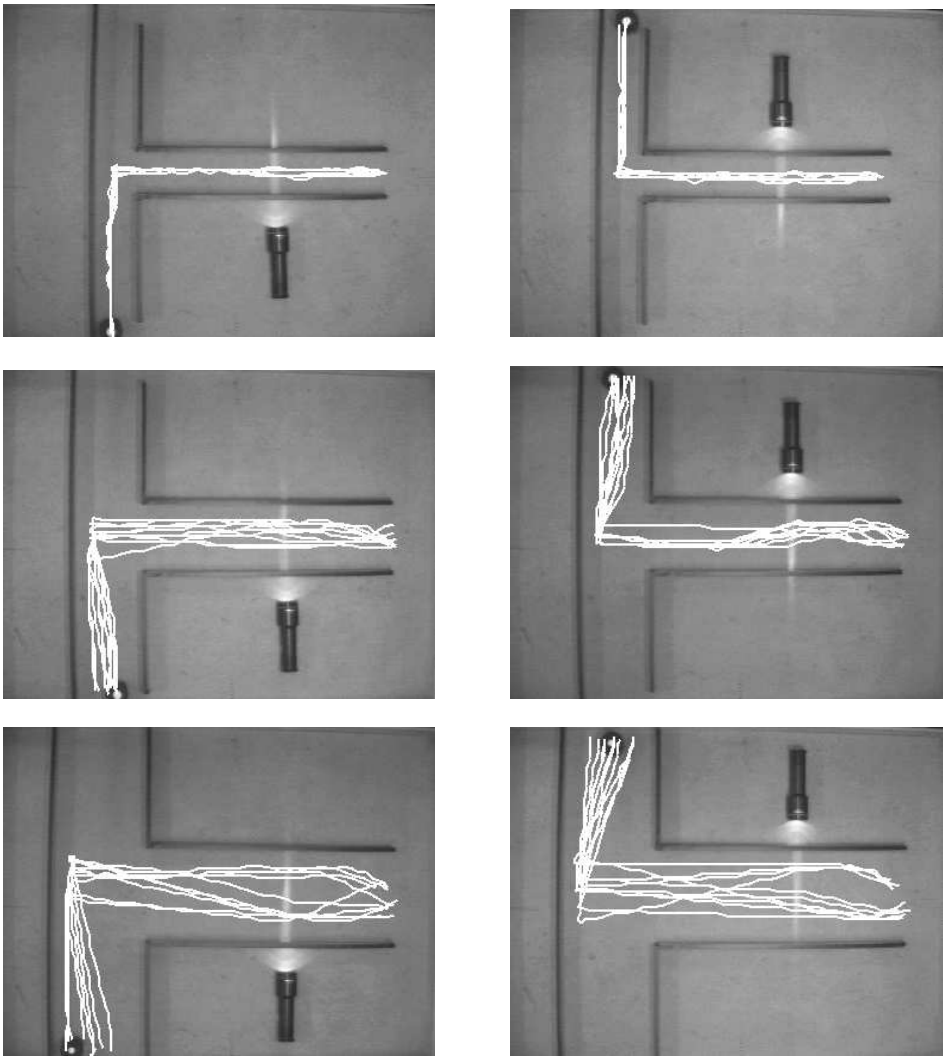

Figure 8: These six pictures together show the paths taken by a Khepera robot in sixty consecutive trials of the control architecture shown in figure 7. These sixty trials were performed in consecutive batches of ten, and each picture shows ten trials for a particular corridor width and torch orientation. 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.

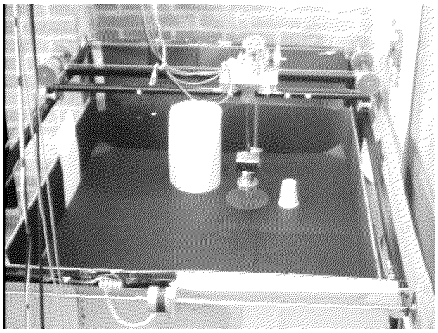

Figure 9: The gantry arena, with obstacles.

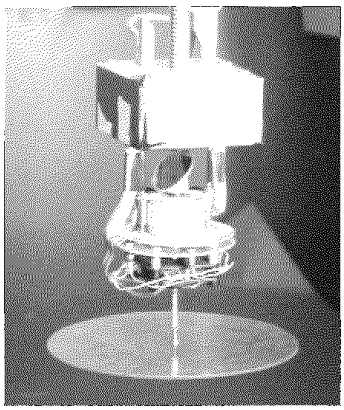

Figure 10: A close up of the gantry robot.

5 A minimal simulation of the gantry robot

This section describes experiments in which neural network controllers were evolved for the gantry robot. The gantry, shown in fig 9, was developed at Sussex University for research into the evolution of visually guided behaviours. It is best thought of as a hardware simulation of a small wheeled mobile robot with a camera placed on top, 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 downwards at a 45^o 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 it to rotate around the vertical axis under computer control and a dedicated vision PC then rotates the image array in software so that 'down' in the picture corresponds to 'down' in reality. The image array available for use by evolving control architectures is therefore equivalent to that produced by a camera pointing outwards along the horizontal component of the mirror's orientation. The frame of the gantry itself is connected to two further stepper motors that together allow the entire robot assembly to move in any horizontal direction within a rectangular arena (see figure 9).

All three stepper motors are controlled by a single board computer (SBC) that is controlled, in turn, by a dedicated brain PC running the control architecture software. The brain PC 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 moves in the appropriate fashion. From the point of view of control architectures running on the brain PC, therefore, the gantry robot behaves exactly as a small wheeled mobile robot, controlled via the SBC, with a camera on top whose image is accessed via the vision PC.

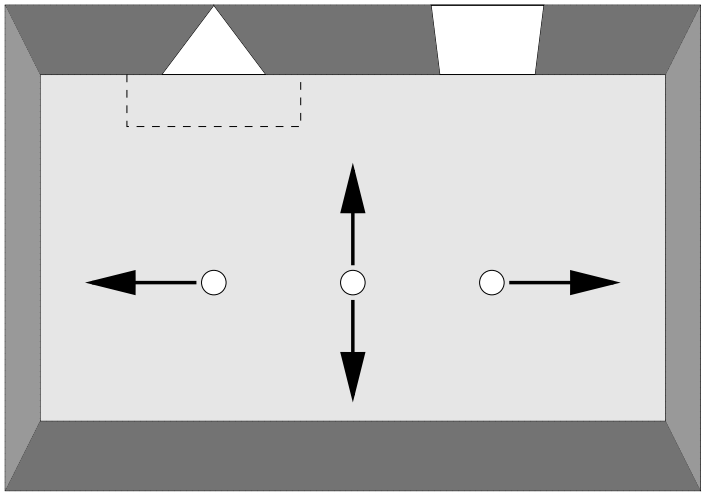

Figure 11: A diagrammatic view of the gantry arena from above showing the four possible starting positions of the gantry robot. The dashed line in front of the triangle marks the area that the gantry must reach in order for a trial to count as a success for testing purposes.

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, 150cm by 100cm, with 22.5cm high walls. Stuck onto one of the long walls were a near-square (20cm wide by 22.5cm high) and an equilateral triangle (20cm wide by 22.5cm 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 towards the triangle as opposed to the square (see Figure 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 around 80% reliable within certain constrained sets of lighting conditions (Husbands, 1997): if the blinds of the laboratory were opened during the day, or if the overhead lighting was not on in the right way, they failed. In order to remedy this sensitivity to differing lighting conditions a set of lamps were strung up above the gantry, each turning on and off at different frequencies, to provide extreme real-world noise for evolving controllers to cope with. 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 (using real-world evolution) that are able to cope with the extra uncertainty that these lights provide.

The control architectures evolved in Harvey et al. (1994) remain amongst the most behaviourally complex that have ever been evolved, even though they were extremely sensitive to lighting conditions and only 80% reliable at the best of times. In addition, the shape discrimination task involves real vision which is an inherently difficult

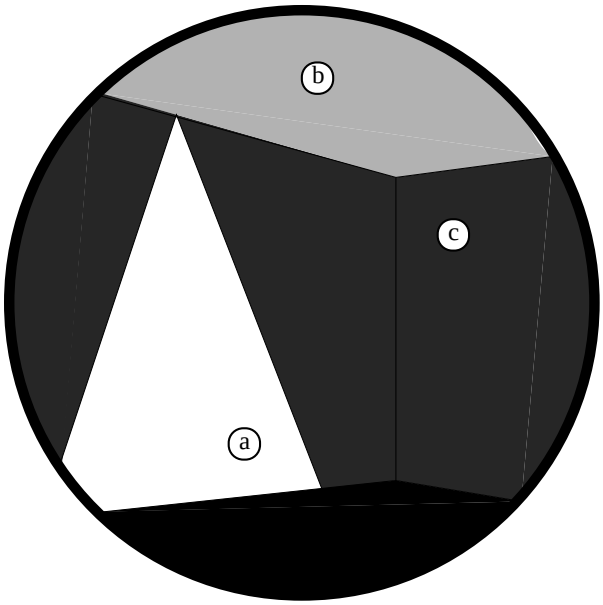

Figure 12: A typical image returned by the camera of the gantry robot. The robot is facing the corner of the arena and the triangle can be seen on the left. The white circles labelled a, b and c are examples of pixels that project onto the triangle, ceiling and wall respectively. Pixel a will return 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 below, each visual input was made up of exactly one pixel whose coordinates within the camera image was genetically determined.

modality to simulate. It was therefore decided that evolving reliably fit control architectures in a simulation built according to the methodology laid out in Section 3, and seeing whether they would be able to perform the shape discrimination task satisfactorily when downloaded into the real world, would provide a good test of the radical envelope of noise hypothesis.

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 below), although both neural networks and the visual morphology of their inputs were again 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 grey-level value of a genetically specified circular sub-region of the camera image, whereas in the experiments reported here, each visual input consisted of the grey-level value of exactly one genetically specified pixel of the camera image (Figure 12). In fact, these are not so different with respect

to a simulation, since 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 both with respect to time, and with respect to the direction of the camera. Even if we know the exact location within the arena that a particular pixel projects onto, there is not that much we can say about exactly what the value of that pixel will be. However, there are a few general things that hold true except in exceptional circumstances: if a pixel projects onto a wall but not onto a shape then it will return a value within the range 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. Since these facts about pixel values within the 'disco light' environment are almost always the case, and since 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 we needed to model.

In fact, the only visual aspect that it was essential to include in the model in order that evolving control architectures were able to perform the shape discrimination task, was the way in which pixels that project *onto the walls of the arena* acquire grey-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 be nothing to do with squares or triangles, there is no point in allowing evolving control architectures to rely on it. This is especially true when one considers the extra modelling required. For example, if the strategy employed by a control architecture that is reliably fit within the simulation depends upon 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 upon 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 grey-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 was again updated at a rate equivalent to ten times a 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 (that the gantry robot is a hardware simulation of) is 15cm 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, momentum plays a significant role since it is a heavy robot which 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 modelling inaccuracies, and can be coped with by reliably fit control architectures that are base set robust (see 3.2). at every iteration, a random deviate in the range ± 0.2 cm per second was added to each wheel speed to approximate the noise inherent in the way the gantry robot moves.

To work out the location in the arena that a particular genetically specified pixel projects onto was done using simple trigonometry. 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^o. In addition there was a finer-grained look-up-table for *tan* containing 200 values, one for every 0.1^o between 0^o and 20^o.

After rotation by the vision PC, 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 around 50^o (see Figure 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 were then 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. Since the coordinates of the robot's position within the arena were always known, and the height of the mirror above the floor of the arena was fixed (around 19.5cm), the exact horizontal and vertical coordinates of the spot that any particular pixel projected onto could be easily worked out. Firstly, 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. Secondly, 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 a pixel projects onto), the course-grained *tan* look-up-table was used, and for calculations of Pz (the vertical coordinate of the point a pixel projects onto), the fine-grained *tan* look-up-table was used. This is because ψ will always be a small angle somewhere between 0^o and 25^o whereas θ can be anything between 0^o and 360^o.

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. Either 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 below. If Pz was less than 0cm or Pz was greater than 22.5cm then the pixel was judged to have projected onto

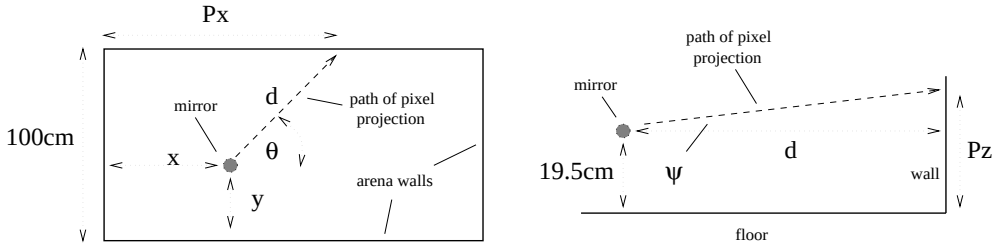

Figure 13: The left-hand picture shows the gantry arena as seen from above; if the horizontal angle at which a pixel projects from the mirror onto the back wall is θ, then $Px = x + \frac{100-y}{tan\theta}$ and $d = \frac{100-y}{sin\theta}$. The right-hand picture shows a cross-section of the gantry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

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

Ensuring that reliably fit controllers were base set exclusive

As reported above, 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, since 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 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 behaviour 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 varied randomly in response to changes in robot-orientation. this meant

that if the robot proceeded in a straight line, or remained still, then pixel vales remained steady. If the robot turned, then pixel values could change. Angular distances between changes in value for any particular pixel averaged 25^o and were uniformly distributed between 0^o and 50^o.

3. Each pixel returned the same random value 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 that ensured they were totally unreliable: random values were returned in a random way between the minimum and maximum values (0 and 15) instead of some sub-interval. 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 that reliably fit controllers were prevented from depending on were the starting position of the robot at the beginning of each trial, and whether the triangle was on the left and the square on the right or vice versa. There were four possible starting positions that varied from trial to trial, see figure 11, and half of the trials that made up the fitness test had the triangle on the left, and half on the right.

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 3.2). 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^o 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 upwards. Because of this:

- A vertical angular offset of between -1^o and -8^o 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 $\pm 3^o$ was produced at the beginning of each trial. This was then 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 were offset by a random amount within the range $\pm$5cm throughout each trial.

The stepper motors move the gantry supporting the robot along rollers using drive-chains. The rollers slide rather than role along their rails (due 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 travelling at a constant speed, and neither accelerates nor breaks evenly in response to motor commands. It will often cease completely half way through a run. In order that reliably fit individuals evolved to cope with these problems:

- The momentum term, m, of equation 5.2 was randomly set at the beginning of each trial to a value between 1 and 4.
- Random offsets of between ± 0.5cm per second 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 mal-aligned 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 non-trivial task independent of which currently available evolutionary techniques are employed. Evolving such behaviours using the simulation described above, furthermore, was even harder, since 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. This is why, although the evolutionary machinery used in Harvey et al. (1994) (control architectures, genetic algorithm, fitness function and so on) was initially reimplemented for the experiments described here in order to provide a direct comparison, it was later 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 experiments described in section 4: weights on links are in the range ± 2 and thresholds are in the range 0 to 1. The activation function of every unit *including* the motor neurons was that of equation 4.3. In addition to a genetically determined number (with a maximum of 3) of connections to each neuron from other neurons in the network, neurons could also receive normalized input, in the range 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 forwards and backwards neurons. The whole network including inputs and outputs, and therefore the whole simulation, was updated at a speed of 10 times per simulated second.

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 SAGA-like principles (Harvey, 1992).

Development took place in a two dimensional space, with the position of each neuron (apart from the four motor neurons, see figure 14) 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, was then alloted as the originator of the link. If no neurons lay within this radius, which was set at around

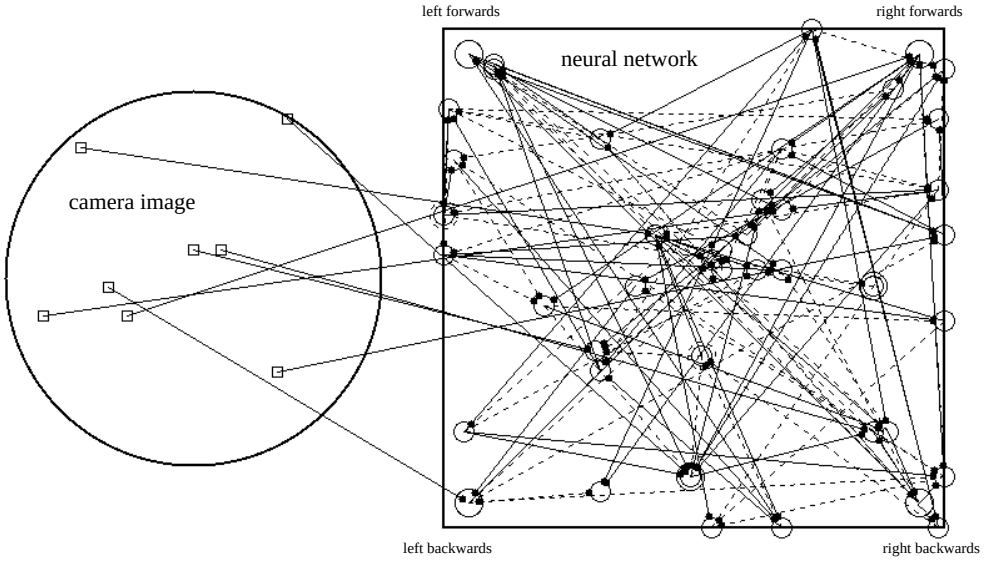

Figure 14: An example of a typical neural network evolved for the triangle/square discrimination task. On the left, the circular camera image, three pixels of which have been genetically specified as inputs to the neural network. On the right, the square box contains the seven neurons of the network.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

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. Since 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 was therefore 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 a neuron received input from a pixel of the camera image or not, 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 number per link. The first two governed which neuron the link originated from by encoding the x and y coordinates of a point within the developmental space. The link was then judged to have originated from the nearest neuron to this point, or not at all if there were no neurons within a range of about 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 and the expected number of mutations per genotype, according to a Poisson distribution, was 1. There was a probability of 0.02 at each offspring event that a random gene would be introduced into the offspring genotype, as well as a probability of 0.02 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 twenty 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 centre of the robot to the centre 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 around 6000 generations of the genetic algorithm (around 12 hours as a single user on a SPARC Ultra), which is the simulated equivalent of $6000 \times 100 \times 8 \times 20 = 96000000$ seconds, or over 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 precedes directly towards it, adjusting its course as it goes.

In order to see whether it would cross the reality gap, the network was downloaded onto the gantry, and tested continuously[4] 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 successfully achieved the task if, by the end of the trial, it was stationed within a rectangular area extending about 10cm either side of the triangle and 15cm out into the arena (see Figure 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 and prevent the motors from jamming) and the lights were temporarily obscured by the author's body. The other failure is harder to account for, since the gantry robot just headed off into a wall under otherwise unremarkable

[4] 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.

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 round to face the triangle. In two out of three of the cases it then locked on to the triangle, and started to move directly towards 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 round to face the triangle for 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 paper, a theoretical investigation was carried out 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 was then 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 will therefore successfully cross the reality gap. It was further argued that if the hypothesis were well-founded, then the creation of minimal simulations would be possible that were easy to build and computationally cheap.

The second part of the paper 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 visually discriminate between two shapes using the Sussex university gantry robot. In both cases, evolved controllers were able to successfully cross the reality gap, exhibiting extremely robust behaviours 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 Mataric and Cliff (1996), the authors suggest that as robots and the behaviours 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 hard to design that the time taken in development will outweigh the time saved in reality. This paper has gone some of the way to showing that for certain types of robots and/or behaviours, at least, this will *not* be the case.

To illustrate this, we offer two examples, one that demonstrates the possibility of creating minimal simulations that support the evolution of complex behaviours, and one which demonstrates the possibility of creating minimal simulations for complex robots. Firstly, consider a slightly extended version of the minimal simulation of the Khepera robot described in Section 4. In this version, instead of a single light signal, the Khepera is presented with a whole series of light signals that together communicate

the solution to a complex maze, with many hundreds of junctions, in morse code. This would be an extremely complicated behaviour by today's standards of what can and cannot be evolved, and yet the minimal simulation remains simple and fast. It *is* therefore possible to build minimal simulations for the evolution of complex behaviours. Secondly, we need only look at the experiments of section 5 to realise that it *is* possible to create minimal simulations for complex robots, or at least robots which employ complex sensory modalities such as vision. The radical envelope of noise hypothesis has yet to be tested on robots with complex motor modalities, such as insect robots.

The point is that whether a minimal simulation is easy to construct and runs fast depends not on the complexity of the behaviour we want to evolve 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 underly the behaviour. Provided these are simple enough, then the behaviour and/or robot 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. Firstly, that results in insect and invertebrate neuroscience suggest that many complex behaviours are often accomplished by way of simple interactions with the environment rather than complicated ones (Collett, 1996; Wehner, 1987; Horridge, 1992). And secondly, that control strategies grounded in complex robot-environment interactions can lead to prohibitively heavy real-time processing requirements (Brooks, 1991): a fact that has fuelled 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 behaviour based robotics of the present day (Chiel, Beer, Quinn, & Espenschied, 1992).

Acknowledgements

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 proof-reading and after hours consultation, Peter Stuer for arguing that it wasn't possible and Tim Smithers for seeing that it might be. Thanks also to the school of COGS itself for the bursary that allows me to undertake this work.

Reference

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.* MIT Press/Bradford Books, Cambridge MA.

Brogan, W. L. (1991). *Modern control theory* (3rd edition). Prentice-Hall.

Brooks, R. (1991). Intelligence without reason. In *Proceedings of the Twelth International Joint Conference on Artificial Intelligence*, pp. 569–95 San Mateo, California. 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 Belew, R. K., & Booker, L. B. (Eds.), *Proceedings of the Fourth Intl. Conf. on Genetic Algorithms, ICGA-91*, pp. 249–256. Morgan Kaufmann.

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

Harvey, I. (1992). Species adaptation genetic algorithms: the basis for a continuing saga. In Varela, F. J., & Bourgine, P. (Eds.), *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pp. 346–354 Cambridge, Massachusetts. M.I.T. Press / Bradford Books.

Harvey, I., Husbands, P., & Cliff, D. (1994). Seeing the light: Artificial evolution, real vision. In Cliff, D., Husbands, P., Meyer, J., & Wilson, S. (Eds.), *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, Vol. 3. 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. (1997). Personal communication..

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*, pp. 139–146. I.E.E.E. Press.

Husbands, P., Harvey, I., & Cliff, D. (1993). An evolutionary approach to situated a.i.. In Sloman, A., Hogg, D., Humphreys, G., Ramsay, A., & Partridge, D. (Eds.), *Prospects for Artificial Intelligence*. I.O.S. Press.

Husbands, P., Harvey, I., Jakobi, N., Thompson, A., & Cliff, D. (1997). Evolutionary robotics. In Back, T., Fogel, D., & Michalewicz, Z. (Eds.), *Handbook of Evolutionary Computation*, chap. G3.7. Oxford University Press.

Jakobi, N. (1996). Encoding scheme issues for open-ended artificial evolution. In Voigt, H.-M., Ebeling, W., Rechenberg, I., & Schwefel, H.-P. (Eds.), *Proceedings of the Fourth International Conferrence on Parallel Problem Solving in Nature*, pp. 52–61 Berlin. Springer-Verlag.

Jakobi, N., Husbands, P., & Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. In Moran, F., Moreno, A., Merelo, J., &

Chacon, P. (Eds.), *Advances in Artificial Life: Proc. 3rd European Conference on Artificial Life*. Springer-Verlag.

K-Team (1993). *Khepera Users Manual*. EPFL,Lausanne.

Mataric, 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 Alliot, J., Lutton, E., Ronald, E., Schoenauer, M., & Snyers, D. (Eds.), *Proceedings of the European Conference on Artificial Evolution*. Springer-Verlag.

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

Nilsson, N. J. (1984). Shakey the robot. Technical note 323, SRI International, Menlo Park, California.

Nolfi, S., Floreano, D., Miglino, O., & Mondada, F. (1994a). How to evolve autonomous robots: Different approaches in evolutionary robotics.. Tech. rep. PCIA-94-03, Department of Cognitive Processes and Artificial Intelligence, 00137, Rome, Italy.

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

Thompson, A. (1995). Evolving electronic robot controllers that exploit hardware resources. In Moran, F., Moreno, A., Merelo, J., & Chacon, P. (Eds.), *Advances in Artificial Life: Proc. 3rd European Conference on Artificial Life*, pp. 640–656. Springer-Verlag, Lecture Notes in Artificial Intelligence 929.

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