Half-baked, Ad-hoc and Noisy: Minimal Simulations for Evolutionary Robotics

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

Abstract

This paper puts forward a theoretical framework and formal language for understanding how simple, fast simulations can be used to artificially evolve controllers for real robots. It begins by putting forward a general set of equations that describe the way in which an agent-environment system changes over time, and analyses what it means for an agent to exhibit a particular behaviour within such a system. A minimally sufficient set of conditions are then formally established under which a controller that reliably displays a particular behaviour in one system will continue to display the same behaviour when transplanted into another system, and the special case of behavioural transference from simulation into reality is evaluated within the context of this theoretical framework. From this, techniques are derived for ensuring that controllers which are reliably fit within a simulation will transfer into reality, and two sets of experiments are briefly described in which controllers that evolved in extremely minimal simulations were able to perform non-trivial and robust behaviours when downloaded onto real robots.

1 Introduction

Several experimenters including [5, 8, 2, 9] 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. As Mataric and Cliff point out in [7], however, if robot controllers evolved in simulation can only be guaranteed to work in reality when a carefully constructed empirically validated simulation is used, then as robots and the behaviours we want to evolve for them become more and more complex, so will the simulations. The worry is that the necessary simulations will be 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. Clearly the main challenge for the simulation approach to evolutionary robotics is to establish that it is possible to evolve control architectures for real robots using simulations that are much simpler than the carefully constructed empirically validated models of the world that Mataric and Cliff talk about.

This paper sets out to meet this challenge. It provides much more than just a few examples of controllers that have evolved in simple simulations transferring into reality however. It introduces a formalism for reasoning about agents performing behaviours in environments, and *derives* a minimal set of conditions for successful behavioural transference. This provides a sound theoretical basis for a practical simulation-building methodology. As we shall see, it *is* possible to build minimal simulations for the evolution of complex behaviours on complex robots that run fast and are easy to build.

In Section 2 the formalism is introduced that describes the way in which the coupled dynamical system that is an agent and its environment changes over time, and in Section 3 we put forward a formal notation for the behaviour of an agent in such a system. Section 4 is the backbone of the paper in which the minimal set of conditions for behavioural transference are formally derived, and in section 5 we show how these conditions can be practically applied and used to build minimal simulations for evolutionary robotics. Section 6 outlines two sets of experiments involving a minimal simulation of a Khepera robot and a minimal simulation of the Sussex university gantry robot. Finally section 7 offers some conclusions.

2 State-space equations for a general agent-environment system

Various researchers have discussed the interactions between an agent and its environment in terms of the interactions between two dynamical systems [10, 1]. This

section introduces a formalism that can be used to describe these interactions, and puts forward general equations for the way in which an agent-environment system changes over time.

The distinction between agent and environment in an agent-environment system is a hard one to define and can be done in a number of ways. For the purposes of this paper, when the agents we are talking about are robots (simulated or real), we draw the line around the agent's controller. This should be kept in mind as the formalism is put forward below. The software or hardware that receives input signals from the sensors and sends output signals to the motors is treated as the agent, and everything else including sensors, actuators and the agent's embodied form is treated as the environment. An agent-environment system is conceptualized as two dynamical systems, the controller and the environment, coupled together through the controller's input and output signals.

The agent's controller

The *internal state* of an agent-environment system is here defined to be the state of the agent's controller at time t. This might consist of the values stored in a set of software registers, or the pattern of activation of an artificial neural network, or the current messages of a classifier system. In general, the internal state of an agent-environment system can be represented by a finite number of variables c_i which can be written as an internal state vector $\vec{c}_t$ where t refers to time.

Now whatever the exact nature of the control architecture employed, the value of $\vec{c}_{t+1}$ will be a function of $\vec{c}_t$ and the current sensor input $\vec{i}_t$. We can therefore write a general equation for the way in which the internal state of an agent-environment system changes through time;

$$\vec{c}_{t+1} = C_1(\vec{i}_t, \vec{c}_t) \tag{1}$$

where the function C_1 defines the exact way in which the internal state of a particular agent-environment system changes through time. Similarly, we may write a general equation for the way in which motor signals $\vec{o}_t$ are generated as a function of the controller's current state:

$$\vec{o}_t = C_2(\vec{c}_t) \tag{2}$$

where the function C_2 defines the exact way in which a particular controller generates motor signals.

It is the functions C_1 and C_2 that determines the dimensionality of the agent-environment system's internal state vector $\vec{c}_t$. Apart from initial conditions, then, it is these functions, and these functions alone that determine the way in which the agent produces motor signals and responds to input signals over time.

The agent's environment

The environment that an agent's controller interacts with might be the physical world or it might be a highly abstract computer model. It is not normally understood to include the agent itself, but for the purposes of this paper, the agent's embodied form, its sensors and its actuators, are as much a part of the environment as objects and ambient light levels. A complete description of the environment will therefore include the agent's position, orientation, motor activity, and so on as well as more objective features. Such a description will from now on be referred to as *the external state* of the agent-environment system. We can represent this external state by a (perhaps infinite) set of state variables e_i, written as an external state vector $\vec{e}_t$ where t refers to time.

Whatever the exact nature of the environment, the value of $\vec{e}_{t+1}$ will be a function of $\vec{e}_t$ and the current motor signals $\vec{o}_t$ issued by the controller. We can therefore write a general equation for the way in which the external state of the environment changes over time:

$$\vec{e}_{t+1} = E_1(\vec{o}_t, \vec{e}_t) \tag{3}$$

Where E_1 defines the exact way in which the external state of a particular agent-environment system changes over time. The way in which sensors produce values that act as inputs to the controller can be described using a similar equation.

$$\vec{i}_t = E_2(\vec{e}_t) \tag{4}$$

Where E_2 defines the exact way in which the agent's sensors interact with the environment to produce input values to the controller.

Agent-environment interaction equations

The total behaviour of an agent-environment system, then, can be described by four equations;

$$\begin{aligned} \vec{e}_{t+1} &= E_1(\vec{o}_t, \vec{e}_t) \qquad & \vec{c}_{t+1} &= C_1(\vec{i}_t, \vec{c}_t) \\ \vec{i}_t &= E_2(\vec{e}_t) \qquad & \vec{o}_t &= C_2(\vec{c}_t) \end{aligned} \tag{5}$$

Such a system, where functions E_1 and E_2 describe the environment and functions C_1 and C_2 describe the controller, will be referred to using the notation $E \rightleftarrows C$.

In the next sections we will look at what it means for an agent to display a particular behaviour within such a system, and go on to ask what the conditions are under which an agent that displays a behaviour in one system will continue to display it in another.

3 What does it mean for an agent to perform a particular behaviour within an agent-environment system

In most cases, whether or not an agent displays a particular behaviour is a function of how the agent *acts* within

its environment[1]. Thus, although the current state of the agent's controller and its sensor inputs are, in general, central to the generation and maintenance of a particular behaviour, it is the controller's output and the effects this has on the external state of the system that determines whether or not the agent is actually performing the behaviour at any given time. To illustrate this, consider the behaviour that consists of driving a car down a road. If someone sits in a car and steers it down the road with a blindfold on, then we will still say they are driving the car (blindfolded) down the road. However, if someone receives visual input as if they were driving the car, but they are not in the driver's seat and are not touching the steering wheel, then we will say they are not driving the car down the road. In general, whether or not an agent is performing a particular behaviour within a particular agent-environment system depends on the way the external state of the system changes through time rather than the internal state.

The way in which the external state of an agent-environment system changes is obviously a function of the agent's motor output, but it is also a function of the environment itself and the ways in which the agent may interact with it. For instance, if a person makes breast-stroke motions with their arms and kicks their legs, then they are producing the correct motor output for swimming. However, it is not until they are immersed in water or something equivalent that they actually *are* swimming[2]. As another example, consider corridor-following behaviour in a small mobile robot. It does not matter what motor output the robot generates, unless the environment includes something that acts like a corridor for the robot, it cannot display the behaviour. The same goes for driving a car down a road (blind-folded or otherwise): without something equivalent to a car or a road, the behaviour is simply not possible.

Any particular behaviour necessitates a base set of environmental features that the agent must be able to interact with in certain ways defined by that behaviour. Unless such features exist, we can never come onto the question of whether the agent is producing appropriate motor output to perform that behaviour. In the swimming example, there must be a base set of features that the agent can interact with *as if it were* immersed in a liquid. In the corridor example, there must be a base set of features that the agent can interact with *as if it were* moving about in a corridor, and in the driving example there must be a base set of features that interact with the sensors and motors of the agent *as if it were* driving a car down a road. The point is that whether or not the agent *actually is* immersed in a liquid, or moving about in a corridor, or driving a car down a road is immaterial. To decide whether or not the environment is capable of supporting the particular behaviour, we just need to know that it contains a base set of features that the agent can interact with *as if it were*.

Each behaviour defines ways in which an agent must *be able* to interact with at least some of its environment. Having established this, we can then go on to decide whether it in fact *does* interact with this portion of the environment in the right way. Whether or not an agent displays a certain behaviour within a particular agent-environment system is contingent upon two facts about the system:

1. The environment must contain a base set of features that the agent can interact with in certain behaviour-defined ways.
2. The way in which the agent actually does interact with these features must constitute an instance of that behaviour.

In the remainder of this section both of these conditions are formalized in turn.

We can define a reduction operator R_e over a particular external state vector $\vec{e}_t$ such that $R_e(\vec{e}_t) = (e_i, e_j, \ldots, e_k)$ is a vector consisting of a particular ordered subset of the variables that make up $\vec{e}_t$. Thus any particular R_e can be thought of as picking out a small portion of the total external state of an agent-environment system and giving it its own reduced state vector. Whether or not an agent-environment system can support a particular behaviour then, is a matter of whether there exists a suitable R_e that can pick out those variables which describe a base set of environmental features that the agent can interact with in the appropriate behaviour defined ways. Such a set of variables will be referred to from now on as the behaviourally defined *base set* of external state variables of the system.

To put this more formally, we say that a behaviour defines a set of functions $\mathbf{B_{env}}$ that each describes one way in which it is acceptable for the behaviourally relevant *base set* of external state variables (picked out by a suitable R_e) to interact with each other and the output of the controller. Given this set of functions, an agent-environment system is capable of supporting the behaviour if and only if

$$\begin{aligned} &\exists R_e \text{ and } \exists B \in \mathbf{B_{env}} \quad s.t. \\ &\quad R_e(\vec{e}_{t+1}) = B(\vec{o}_t, R_e(\vec{e}_t)) \qquad \forall \vec{o}_t, \vec{e}_t \end{aligned} \tag{6}$$

In other words $\mathbf{B_{env}}$ defines all the possible ways B in which the agent can interact with some reduced external state vector $R_e(\vec{e}_t)$ if it is to perform the behaviour. If we cannot find such a base set of external state variables that the agent can interact with in one of the behaviourally acceptable ways, then the agent-environment system is not capable of supporting the behaviour.

[1] There are certain behaviours, such as dreaming or 'having an idea', that this is not true of, but since there is no difference between simulation and reality for such behaviours, they shall not be considered here.

[2] Thanks to Joe Faith for this example.

If we *can* find a suitable R_e such that equation 6 is satisfied, then whether or not the agent actually does perform the behaviour in a given time interval is a function of how the variables that R_e picks out change over time. We can describe the way in which $R_e(\vec{e}_t)$ changes over time on a particular occasion as an ordered set of the states it passes through in a time interval $t = 0, 1, 2, \ldots, \tau$. This ordered set shall be referred to as the state-space trajectory T_{R_e}, where

$$T_{R_e} = \{R_e(\vec{e}_t) : t = 0, 1, 2, \ldots, \tau\}$$

Whether or not a particular state-space trajectory T_{R_e} does or does not constitute an instance of the behaviour is a behaviourally defined function of that trajectory. We shall refer to this function as B_{act} for each behaviour. Thus, in addition to the set of functions $\mathbf{B_{env}}$, each behaviour also defines and is defined by a function B_{act} that return true or false such that

$$B_{act}(T_{R_e}) \Leftrightarrow \text{the agent exhibits the behaviour} \quad (7)$$

Putting equations 6 and 7 together, we can say that an agent *reliably* performs a particular behaviour, defined by $\mathbf{B_{env}}$ and B_{act}, within an environment with external state vector $\vec{e}_t$ if and only if

$$\begin{array}{l} \exists R_e \text{ and } \exists B \in \mathbf{B_{env}} \quad s.t. \\ \quad R_e(\vec{e}_{t+1}) = B(\vec{o}_t, R_e(\vec{e}_t)) \quad \forall \vec{o}_t, \vec{e}_t \\ \text{and} \\ \quad B_{act}(T_{R_e}) \forall T_{R_e} \end{array}$$

4 What are the conditions under which an agent's behaviour in one system implies its reliable behaviour in another?

In this section we formally derive a minimal set of conditions that must be true of a controller and a simulation if the controller's reliable behaviour in simulation is to guarantee its reliable behaviour in reality. The derivation precedes in four stages. In the first stage we provide a formal proof that if two agent-environment systems are identical in every respect and deterministic, then any behaviour that the agent reliably performs in one system, will also be reliably performed in the other. This proof is extended over the next three stages as we examine pairs of agent-environment systems that bear looser and looser relationships to each other. In the final stage, we arrive at a pair of agent-environment systems that bear the same relationship to each other as simulations of robots bear to real robots. Since at each stage we put forward necessary conditions that must be fulfilled by the agent's controller if the proof is to remain valid, by the final stage we are left with a minimal set of conditions that must be fulfilled by an agent's controller and a simulation if the controller's reliable behaviour in simulation is to guarantee its reliable behaviour in reality.

Before we go on to derive conditions for anything, however, we must formally define what it is we want to derive conditions for. Without this, we won't know what it looks like when we've done it. From Section 2, we can write agent-environment interaction equations for two systems, $E \rightleftarrows C$ and $S \rightleftarrows C$. Note however that the controller functions C_1 and C_2 are the same for both systems. To avoid confusion, the symbols for each controller's input, output, and internal state vectors include superscript symbols corresponding to the names of their external state vectors.

$$\begin{array}{ll} \vec{e}_{t+1} = E_1(\vec{o}^e_t, \vec{e}_t) & \vec{c}^e_{t+1} = C_1(\vec{i}^e_t, \vec{c}^e_t) \\ \vec{i}^e_t = E_2(\vec{e}_t) & \vec{o}^e_t = C_2(\vec{c}^e_t) \end{array} \quad (8)$$

and

$$\begin{array}{ll} \vec{s}_{t+1} = S_1(\vec{o}^s_t, \vec{s}_t) & \vec{c}^s_{t+1} = C_1(\vec{i}^s_t, \vec{c}^s_t) \\ \vec{i}^s_t = S_2(\vec{s}_t) & \vec{o}^s_t = C_2(\vec{c}^s_t) \end{array} \quad (9)$$

The question that this section is concerned with is as follows: given that both $S \rightleftarrows C$ and $E \rightleftarrows C$ are capable of supporting a particular behaviour defined by $\mathbf{B_{env}}$ and B_{act}, and that the environments of both systems bear certain relationships to each other, what are the conditions that must be satisfied in order that the fact that the agent reliably performs the behaviour in system $S \rightleftarrows C$ implies that it will also reliably perform the behaviour in $E \rightleftarrows C$?

To put this more formally, given that

$$\begin{array}{l} \exists R_s \text{ and } \exists B \in \mathbf{B_{env}} \quad s.t. \\ \quad R_s(\vec{s}_{t+1}) = B(\vec{o}^s_t, R_s(\vec{s}_t)) \qquad \forall \vec{o}^s_t, \vec{s}_t \end{array}$$

and

$$\begin{array}{l} \exists R_e \text{ and } \exists B \in \mathbf{B_{env}} \quad s.t. \\ \quad R_e(\vec{e}_{t+1}) = B(\vec{o}^e_t, R_e(\vec{e}_t)) \qquad \forall \vec{o}^e_t, \vec{e}_t \end{array}$$

and

$$B_{act}(T_{R_s}) \quad \forall T_{R_s}$$

what are the conditions that must be true of the functions and constants of equations 8 and 9 in order that $B_{act}(T_{R_e}) \forall T_{R_e}$?

4.1 Both systems are deterministic and identical

If the two agent-environment interaction systems $S \rightleftarrows C$ and $E \rightleftarrows C$ are identical, then intuitively we expect every controller that can reliably perform a particular behaviour in one system to reliably perform it in the other. We shall formally prove this here.

The identity of the two systems implies that all the functions of equations 9 are identical to their corresponding functions in equations 8, and that there is a one to one mapping between the constants of both systems. In the case where both systems are deterministic (we will come onto the non-deterministic case in section 4.3), we can express these relations formally as:

$$\begin{aligned} & \vec{e}_t = \vec{s}_t \quad \text{and} \quad \vec{o}^{\,e}_t = \vec{o}^{\,s}_t \qquad (10) \\ \Rightarrow \quad & E_1(\vec{e}_t, \vec{o}^{\,e}_t) = S_1(\vec{s}_t, \vec{o}^{\,s}_t) \quad \forall \vec{s}_t, \vec{o}^{\,e}_t, \vec{o}^{\,s}_t, \vec{e}_t \end{aligned}$$

and

$$\begin{aligned} & \vec{e}_t = \vec{s}_t \qquad (11) \\ & \quad \Rightarrow \quad E_2(\vec{e}_t) = S_2(\vec{s}_t) \quad \forall \vec{s}_t, \vec{e}_t \end{aligned}$$

for the identity of the functions, and for the one to one mapping between the constants:

$$\forall \vec{s}_t \quad \exists \vec{e}_t \quad s.t. \quad \vec{e}_t = \vec{s}_t \qquad (12)$$

$$\forall \vec{e}_t \quad \exists \vec{s}_t \quad s.t. \quad \vec{s}_t = \vec{e}_t \qquad (13)$$

Note that we cannot say the same thing if either system incorporates a non-deterministic element. We will now prove that if these relations hold, then if a controller reliably performs a particular behaviour in $S \rightleftarrows C$ then it must also reliably perform the behaviour in $E \rightleftarrows C$:

$$\begin{aligned} & \vec{e}_t = \vec{s}_t \\ \Rightarrow \quad & \vec{i}^{\,e}_t = E_2(\vec{e}_t) = S_2(\vec{s}_t) = \vec{i}^{\,s}_t \quad \text{(from 11)} \\ \Rightarrow \quad & \vec{c}^{\,e}_{t+1} = C_1(\vec{i}^{\,e}_t, \vec{c}^{\,e}_t) = C_1(\vec{i}^{\,s}_t, \vec{c}^{\,s}_t) = \vec{c}^{\,s}_{t+1} \\ \Rightarrow \quad & \vec{o}^{\,e}_t = C_2(\vec{c}^{\,e}_t) = C_2(\vec{c}^{\,s}_t) = \vec{o}^{\,s}_t \\ \Rightarrow \quad & E_1(\vec{o}^{\,e}_t, \vec{e}_t) = S_1(\vec{o}^{\,s}_t, \vec{s}_t) \quad \text{(from 10)} \\ \Rightarrow \quad & \vec{e}_{t+1} = \vec{s}_{t+1} \end{aligned}$$

$$\begin{aligned} \therefore \quad & \forall T_s \ \forall T_e \quad (\ \vec{e}_0 = \vec{s}_0 \quad \Rightarrow \quad T_e = T_s \ ) \\ \therefore \quad & \forall T_e \ \exists T_s \quad s.t. \quad T_s = T_e \quad \text{(from 13)} \\ \Rightarrow \quad & \forall T_{R_e} \quad \exists T_{R_s} \quad s.t. \quad T_{R_s} = T_{R_e} \end{aligned}$$

$$\therefore \quad B_{act}(T_{R_s}) \ \forall T_{R_s} \quad \Rightarrow \quad B_{act}(T_{R_e}) \ \forall T_{R_e}$$

4.2 Both systems are deterministic but only identical with respect to their controllers how they interact with the base set of external state variables

The proof given in subsection 4.1 above involves the one situation in which it is intuitively evident that *any* agent that reliably performs a behaviour in one system will also perform it in the other: namely the situation in which both systems are deterministic and identical. In this subsection we examine the situation in which our two agent-environment interaction systems $E \rightleftarrows C$ and $S \rightleftarrows C$ are still deterministic, but only identical with respect to their controllers and the way in which these controllers interact with the behaviourally defined base set of external state variables. In this situation, as we shall see, it is only possible to prove that the controller's reliable behaviour in one system implies its reliable behaviour in the other for agent controllers that satisfy particular conditions. These conditions are given below.

Crucial to the proof of the previous section is the part that shows that $(\vec{e}_t = \vec{s}_t) \ \Rightarrow \ (\vec{e}_{t+1} = \vec{s}_{t+1})$. If we can find the conditions under which $(R_e(\vec{e}_t) = R_s(\vec{s}_t)) \ \Rightarrow \ (R_e(\vec{e}_{t+1}) = R_s(\vec{s}_{t+1}))$, for the situation under consideration here, then the rest follows easily. The problem is that if the environment functions of $E \rightleftarrows C$ and $S \rightleftarrows C$ are only identical with respect to their behaviourally defined base sets of external variables, then the equivalents of equations 10, 12 and 13 are still valid, but the equivalent of equation 11 is not. In symbols,

$$\begin{aligned} & R_e(\vec{e}_t) = R_s(\vec{s}_t) \quad \text{and} \quad \vec{o}^{\,e}_t = \vec{o}^{\,s}_t \qquad (14) \\ \Rightarrow \quad & R_e\big(E_1(\vec{e}_t, \vec{o}^{\,e}_t)\big) = R_s\big(S_1(\vec{s}_t, \vec{o}^{\,s}_t)\big) \quad \forall \vec{s}_t, \vec{o}^{\,e}_t, \vec{o}^{\,s}_t, \vec{e}_t \end{aligned}$$

and

$$\forall R_s(\vec{s}_t) \quad \exists R_e(\vec{e}_t) \quad s.t. \quad R_e(\vec{e}_t) = R_s(\vec{s}_t) \qquad (15)$$

$$\forall R_e(\vec{e}_t) \quad \exists R_s(\vec{s}_t) \quad s.t. \quad R_s(\vec{s}_t) = R_e(\vec{e}_t) \qquad (16)$$

but

$$R_e(\vec{e}_t) = R_s(\vec{s}_t) \quad \not\Rightarrow \quad E_2(\vec{e}_t) = S_2(\vec{s}_t) \quad \forall \vec{s}_t, \vec{e}_t$$

since E_2 and S_2 are functions of the entire external state vectors of both systems. However, we know that the effects of the base sets of external state variables on the values of E_2 and S_2 are the same by definition of the relationship between the two systems. We may therefore place certain restrictions on the controller function C_1, and the way in which it responds to input, that allow us to write an equation for the relationship between E_2 and S_2 that can be used in the proof given below but that does not require that $\vec{e}_t = \vec{s}_t$.

In words, provided that the aspects of the controller input which can affect C_1 derive exclusively from the values of the base set of external variables *and those values alone*, then if those values are the same in both systems, C_1 will respond in the same way in both systems to the same internal state. More formally, substituting equation 4 into equation 1, provided that

$$\begin{aligned} & \vec{R}_s(s_i) = \vec{R}_s(s_j) \qquad (17) \\ \Rightarrow \quad & C_1\big(S_2(\vec{s}_i), c^s_t\big) = C_1\big(S_2(\vec{s}_j), c^s_t\big) \quad \forall \vec{s}_i, \vec{s}_j \end{aligned}$$

then the fact that E_2 and S_2 are defined to be identical with respect to the base set of external state variables

means that

$$\begin{aligned} & \vec{R}_e(e_t) = \vec{R}_s(s_t) \qquad (18) \\ \Rightarrow \quad & C_1\big(E_2(\vec{e}_t), c_t^e\big) = C_1\big(S_2(\vec{s}_t), c_t^s\big) \quad \forall \vec{s}_t, \vec{e}_t \end{aligned}$$

Equation 17 is a condition on the controller that must be true if equation 18 is to work. Simply put, it requires that the internal state of the system depends exclusively on those features of the environment that are described by the base set of external state variables. We shall refer to controllers that fulfill this condition for a particular behaviourally defined base set of external variables as being *base set exclusive*. For controllers that are base set exclusive, we shall now prove that the agent's reliable behaviour in $S \rightleftarrows C$ implies its reliable behaviour in $E \rightleftarrows C$.

$$\begin{aligned} & R_e(\vec{e}_t) = R_s(\vec{s}_t) \\ \Rightarrow \quad & C_1\big(E_2(\vec{e}_t), c_t^e\big) = C_1\big(S_2(\vec{s}_t), c_t^s\big) \quad \text{(from 18)} \\ \Rightarrow \quad & \vec{o}_t^{\,e} = C_2(\vec{c}_t^{\,e}) = C_2(\vec{c}_t^{\,s}) = \vec{o}_t^{\,s} \\ \Rightarrow \quad & R_e\big(E_1(\vec{o}_t^{\,e}, \vec{e}_t)\big) = R_s\big(S_1(\vec{o}_t^{\,s}, \vec{s}_t)\big) \quad \text{(from 14)} \\ \Rightarrow \quad & R_e(\vec{e}_{t+1}) = R_s(\vec{s}_{t+1}) \end{aligned}$$

$$\begin{aligned} \therefore \quad & \forall T_{R_s}\ \forall T_{R_e} \quad (R_e(\vec{e}_0) = R_s(\vec{s}_0) \quad \Rightarrow \quad T_{R_e} = T_{R_s}) \\ \therefore \quad & \forall T_{R_e}\ \exists T_{R_s} \quad s.t. \quad T_{R_s} = T_{R_e} \quad \text{(from 16)} \end{aligned}$$

$$\therefore \quad B_{act}(T_{R_s}) \quad \forall T_{R_s} \quad \Rightarrow \quad B_{act}(T_{R_e}) \quad \forall T_{R_e}$$

4.3 Both systems are non-deterministic and only identical with respect to their controllers and how they interact with the base set of external state variables

In this subsection we consider the situation in which two agent-environment systems (which we shall again call $E \rightleftarrows C$ and $S \rightleftarrows C$) are identical with respect to their controllers and the behaviourally defined base set of external state variables and the functions that affect them, but this time the functions E_1, E_2, S_1 and S_2 are *non-deterministic*. The problem here is that even if two functions are identical they will still return different values so we cannot use equations 14 and 18 to describe how the two systems change over time. However, if two non-deterministic functions are identical, we can express at least some of what this involves by stating that *whatever* the value that one function comes up with, it is *possible* that the other function comes up with the same value. Although this is a much weaker relation than identity, as we shall see, it is in fact all we need. The relationships that the functions must have to each other if the proof given below is to work are therefore given by

$$\begin{aligned} & R_e(\vec{e}_t) = R_s(\vec{s}_t) \quad \text{and} \quad \vec{o}_t^{\,e} = \vec{o}_t^{\,s} \qquad (19) \\ \Rightarrow \quad & \forall E_1(\vec{e}_t, \vec{o}_t^{\,e}) \quad \exists S_1(\vec{s}_t, \vec{o}_t^{\,s}) \qquad s.t. \\ & R_e\big(E_1(\vec{e}_t, \vec{o}_t^{\,e})\big) = R_s\big(S_1(\vec{s}_t, \vec{o}_t^{\,s})\big) \quad \forall \vec{s}_t, \vec{o}_t^{\,e}, \vec{o}_t^{\,s}, \vec{e}_t \end{aligned}$$

and

$$\begin{aligned} & \vec{R}_e(e_t) = \vec{R}_s(s_t) \qquad (20) \\ \Rightarrow \quad & \forall E_2(\vec{e}_t) \quad \exists S_2(\vec{s}_t) \qquad s.t. \\ & C_1\big(E_2(\vec{e}_t), c_t^e\big) = C_1\big(S_2(\vec{s}_t), c_t^s\big) \quad \forall \vec{s}_t, \vec{e}_t \end{aligned}$$

where equation 20 is only guaranteed to work for those controllers that are base set exclusive i.e. that satisfy equation 17. As for the mapping between constants, it is the same as before. The difference being that we realise that in fact we only require a mapping from the external state of $E \rightleftarrows C$ onto the external state of $S \rightleftarrows C$, thus

$$\forall R_e(\vec{e}_t) \quad \exists R_s(\vec{s}_t) \quad s.t. \quad R_s(\vec{s}_t) = R_e(\vec{e}_t) \qquad (21)$$

Now even if these three relationships hold we can not prove that $(R_e(\vec{e}_t) = R_s(\vec{s}_t)) \Rightarrow (R_E(\vec{e}_{t+1}) = R_s(\vec{s}_{t+1}))$. However, we will now prove that if $R_e(\vec{e}_t) = R_s(\vec{s}_t)$ then it is always *possible* that $R_E(\vec{e}_{t+1}) = R_s(\vec{s}_{t+1})$, and as we shall see, this is good enough.

$$\begin{aligned} & R_e(\vec{e}_t) = R_s(\vec{s}_t) \qquad (22) \\ \Rightarrow \quad & \forall E_2(\vec{e}_t) \quad \exists S_2(\vec{s}_t) \quad s.t. \\ & C_1\big(E_2(\vec{e}_t), c_t^e\big) = C_1\big(S_2(\vec{s}_t), c_t^s\big) \quad \text{(from 20)} \\ \Rightarrow \quad & \forall E_2(\vec{e}_t) \quad \exists S_2(\vec{s}_t) \quad s.t. \\ & \vec{o}_t^{\,e} = C_2(\vec{c}_t^{\,e}) = C_2(\vec{c}_t^{\,s}) = \vec{o}_t^{\,s} \\ \Rightarrow \quad & \forall E_2(\vec{e}_t), E_1(\vec{o}_t^{\,e}, \vec{e}_t) \quad \exists S_2(\vec{s}_t), S_1(\vec{o}_t^{\,s}, \vec{s}_t) \quad s.t. \\ & R_e\big(E_1(\vec{o}_t^{\,e}, \vec{e}_t)\big) = R_s\big(S_1(\vec{o}_t^{\,s}, \vec{s}_t)\big) \quad \text{(from 19)} \\ \Rightarrow \quad & \forall E_2(\vec{e}_t), E_1(\vec{o}_t^{\,e}, \vec{e}_t) \quad \exists S_2(\vec{s}_t), S_1(\vec{o}_t^{\,s}, \vec{s}_t) \quad s.t. \\ & R_e(\vec{e}_{t+1}) = R_s(\vec{s}_{t+1}) \end{aligned}$$

In words, this proves that if $R_e(\vec{e}_t) = R_s(\vec{s}_t)$, then whatever the values of E_1 and E_2, it is *possible* that S_1 and S_2 produce identical values, and therefore that $R_e(\vec{e}_{t+1}) = R_s(\vec{s}_{t+1})$. The reason why this is important is that when we say $\forall T_{R_s}$, we are talking about all *possible* trajectories of R_s through time, and when we say $\exists T_{R_s}$ we are saying that there exists a *possible* trajectory of R_s through time. The results of equations 22 allow us to say therefore that provided the controller is base set exclusive:

$$\begin{aligned} & \forall T_{R_e} \quad \big(R_e(\vec{e}_0) = R_s(\vec{s}_0)\big) \quad \Rightarrow \quad \big(\exists T_{R_s} \quad s.t. \quad T_{R_e} = T_{R_s}\big) \\ \therefore \quad & \forall T_{R_e}\ \exists T_{R_s} \qquad s.t. \quad T_{R_s} = T_{R_e} \quad \text{(from 21)} \end{aligned}$$

$$\therefore \quad B_{act}(T_{R_s}) \quad \forall T_{R_s} \quad \Rightarrow \quad B_{act}(T_{R_e}) \quad \forall T_{R_e}$$

4.4 *Both systems are non-deterministic and only identical with respect to their controllers*

In this subsection we examine the situation in which not only are our two agent-environment systems (which we shall again call $E \rightleftarrows C$ and $S \rightleftarrows C$) non-deterministic, but they are also no more than *similar* with respect to their behaviourally defined base sets of external variables and the functions that affect them. The controllers in both systems are again identical. A little reflection reveals that this is in fact exactly the same relationship as exists between a simulation and reality: the real world and a (non-naive) simulation are both non-deterministic, and it is never possible to model any set of features of the real world with 100% accuracy. If we can find conditions on the controller, therefore, such that its reliable behaviour in $S \rightleftarrows C$ implies its reliable behaviour in $E \rightleftarrows C$ for the situation under consideration here, then these are the conditions that must be fulfilled if a controller's reliable behaviour in simulation is to guarantee its reliable behaviour in reality.

The problem is that we cannot say anything definite about the relations between the functions E_1, E_2, S_1 and S_1 without an exact knowledge of exactly how different or similar they are with respect to their base sets of external state variables. Imagine, however, that we have at least *some* idea of just how similar or different the two systems are. In other words, even though we don't know the exact functions S_1' and S_2' that make equations 23 and 24 work, we are still able to define two *sets* of functions centred around S_1 and S_2 which we shall call $\mathbf{U}(S_1)$ and $\mathbf{V}(S_2)$ such that we know that $\exists S_1' \in \mathbf{U}(S_1)$ and $\exists S_2' \in \mathbf{V}(S_2)$ such that

$$\begin{aligned} & R_e(\vec{e}_t) = R_s(\vec{s}_t) \quad \text{and} \quad \vec{o}^{\,e}_t = \vec{o}^{\,s}_t \qquad (23)\\ \Rightarrow \quad & \forall E_1(\vec{e}_t, \vec{o}^{\,e}_t) \quad \exists S_1'(\vec{s}_t, \vec{o}^{\,s}_t) \qquad s.t. \\ & R_e\big(E_1(\vec{e}_t, \vec{o}^{\,e}_t)\big) = R_s\big(S_1'(\vec{s}_t, \vec{o}^{\,s}_t)\big) \quad \forall \vec{s}_t, \vec{o}^{\,e}_t, \vec{o}^{\,s}_t, \vec{e}_t \end{aligned}$$

and

$$\begin{aligned} & \vec{R}_e(e_t) = \vec{R}_s(s_t) \qquad (24)\\ \Rightarrow \quad & \forall E_2(\vec{e}_t) \quad \exists S_2'(\vec{s}_t) \qquad s.t. \\ & C_1\big(E_2(\vec{e}_t), c^e_t\big) = C_1\big(S_2'(\vec{s}_t), c^s_t\big) \quad \forall \vec{s}_t, \vec{e}_t \end{aligned}$$

Where equation 24 is only guaranteed to work for controllers that are base set exclusive. In this situation, however, in order that equation 24 works whatever the nature of function S_2' turns out to be, we need to define the property of being base set exclusive in a more general way

$$\begin{aligned} & \forall S_2' \ \in \ \mathbf{V}(S_2) \qquad (25)\\ & \vec{R}_s(s_i) = \vec{R}_s(s_j) \\ \Rightarrow \quad & C_1\big(S_2'(\vec{s}_i), c^s_t\big) = C_1\big(S_2'(\vec{s}_j), c^s_t\big) \quad \forall \vec{s}_i, \vec{s}_j \end{aligned}$$

The mapping between constants remains the same as that of equation 21:

$$\forall R_e(\vec{e}_t) \quad \exists R_s(\vec{s}_t) \quad s.t. \quad R_s(\vec{s}_t) = R_e(\vec{e}_t) \qquad (26)$$

Given these relations, is there a way of adapting the proof of section 4.3, perhaps by placing extra conditions on the controller, so that its reliable behaviour in $S \rightleftarrows C$ guarantees its reliable behaviour in $E \rightleftarrows C$? The answer unfortunately is no, since there are trajectories of $R_e(\vec{e}_t)$ in $E \rightleftarrows C$ that are not possible in $S \rightleftarrows C$ and visa versa. However if we can define functions $\mathbf{U}(S_1)$ and $\mathbf{V}(S_2)$ such that we know that they contain the appropriate S_1' and S_2' that will make equations 23 and 24 hold, then provided the controller is base set exclusive and can perform reliably in all possible $S' \rightleftarrows C$, it will also perform reliably in $E \rightleftarrows C$.

More formally, if we can define appropriate functions $\mathbf{U}(S_1)$ and $\mathbf{V}(S_2)$ then for those controllers that are base set exclusive,

$$\exists S_1' \in \mathbf{U}(S_1) \quad \text{and} \quad S_2' \in \mathbf{V}(S_2)$$

such that the relations 23 and 24 hold, and

$$\begin{aligned} & R_e(\vec{e}_t) = R_s(\vec{s}_t) \\ \Rightarrow \quad & \forall E_2(\vec{e}_t) \quad \exists S_2'(\vec{s}_t) \quad s.t. \\ & C_2\big(E_2(\vec{e}_t), c^e_t\big) = C_2\big(S_2'(\vec{s}_t), c^s_t\big) \quad \text{(from 24)} \\ \Rightarrow \quad & \forall E_2(\vec{e}_t) \quad \exists S_2'(\vec{s}_t) \quad s.t. \\ & \vec{o}^{\,e}_t = C_2(\vec{c}^{\,e}_t) = C_2(\vec{c}^{\,s}_t) = \vec{o}^{\,s}_t \\ \Rightarrow \quad & \forall E_2(\vec{e}_t), E_1(\vec{o}^{\,e}_t, \vec{e}_t) \quad \exists S_2'(\vec{s}_t), S_1'(\vec{o}^{\,s}_t, \vec{s}_t) \quad s.t. \\ & R_e\big(E_1(\vec{o}^{\,e}_t, \vec{e}_t)\big) = R_s\big(S_1'(\vec{o}^{\,s}_t, \vec{s}_t)\big) \quad \text{(from 23)} \\ \Rightarrow \quad & \forall E_2(\vec{e}_t), E_1(\vec{o}^{\,e}_t, \vec{e}_t) \quad \exists S_2'(\vec{s}_t), S_1'(\vec{o}^{\,s}_t, \vec{s}_t) \quad s.t. \\ & R_e(\vec{e}_{t+1}) = R_s(\vec{s}_{t+1}) \end{aligned}$$

This means that

$$\begin{aligned} & \exists S_1' \in \mathbf{U}(S_1) \quad \exists S_2' \in \mathbf{V}(S_2) \qquad s.t. \\ \forall T_{R_e} \quad & \big(R_e(\vec{e}_0) = R_{s\prime}(\vec{s}_0)\big) \quad \Rightarrow \quad \big(\exists T_{R_s} \quad s.t. \quad T_{R_e} = T_{R_s}\big) \end{aligned}$$

and therefore that

$$\begin{aligned} & \exists S_1' \in \mathbf{U}(S_1) \quad \exists S_2' \in \mathbf{V}(S_2) \quad s.t. \\ & \forall T_{R_e} \quad \exists T_{R_s} \quad s.t. \quad T_{R_e} = T_{R_s} \end{aligned}$$

and therefore that

$$\begin{aligned} & \exists S_1' \in \mathbf{U}(S_1) \quad \exists S_2' \in \mathbf{V}(S_2) \quad s.t. \qquad (27)\\ & B_{act}(T_{R_s})\ \forall T_{R_s} \quad \Rightarrow \quad B_{act}(T_{R_e})\ \forall T_{R_e} \end{aligned}$$

So the condition that the controller reliably performs the behaviour in $S \rightleftarrows C$ is not strong enough to guarantee that it will reliably perform the behaviour in $E \rightleftarrows C$. In order to guarantee this we need to know that the

controller reliably performs the behaviour in all $S' \rightleftarrows C$, or more formally that

$$\forall S'_1 \in \mathbf{U}(S_1) \quad \forall S'_2 \in \mathbf{V}(S_2) \\ B_{act}(T_{R_s}) \; \forall T_{R_s}$$

If this is the case then equation 27 follows.

This, then, is another condition that we must place on the agent's controller. A controller that fulfills this condition will be referred to as being *base set robust*. Not only must it reliably perform the behaviour in the agent-environment system $S \rightleftarrows C$, it must also reliably perform the behaviour in a whole set of associated agent-environment systems $S' \rightleftarrows C$ defined by suitable functions $\mathbf{U}$ and $\mathbf{V}$. Only then can we guarantee that the controller will also reliably perform the behaviour in $E \rightleftarrows C$.

4.5 Minimal conditions for crossing the reality gap

We are now in a position where we can sum up the last section and put forward a minimal set of conditions for the transfer of a behaviour defined by $\mathbf{B_{env}}$ and B_{act} from a simulation $S \rightleftarrows C$ to reality $E \rightleftarrows C$.

1. Both systems must be able to support the behaviour i.e.

$$\exists R_s \text{ and } \exists B \in \mathbf{B_{env}} \quad s.t. \\ R_s(\vec{s}_{t+1}) = B(\vec{o}^s_t, R_s(\vec{s}_t)) \qquad \forall \vec{o}^s_t, \vec{s}_t \tag{28}$$

and

$$\exists R_e \text{ and } \exists B \in \mathbf{B_{env}} \quad s.t. \\ R_e(\vec{e}_{t+1}) = B(\vec{o}^e_t, R_e(\vec{e}_t)) \qquad \forall \vec{o}^e_t, \vec{e}_t \tag{29}$$

2. There must be a mapping from the base set of external variables of the real world onto the base set of external variables of the simulation i.e.

$$\forall R_e(\vec{e}_t) \quad \exists R_s(\vec{s}_t) \quad s.t. \quad R_s(\vec{s}_t) = R_e(\vec{e}_t) \tag{30}$$

3. We must be able to find two sets of functions $\mathbf{U}(S_1)$ and $\mathbf{V}(S_2)$ such that the controller is *base set exclusive* and *base set robust* i.e. $\forall S'_1 \in \mathbf{U}(S_1)$ and $\forall S'_2 \in \mathbf{V}(S_1)$,

$$\vec{R}_s(s_i) = \vec{R}_s(s_j) \\ \Rightarrow \quad C_1\big(S'_2(\vec{s}_i), c^s_t\big) = C_1\big(S'_2(\vec{s}_j), c^s_t\big) \quad \forall \vec{s}_i, \vec{s}_j \tag{31}$$

and

$$B_{act}(T_{R_s}) \; \forall T_{R_s} \tag{32}$$

and such that $\exists S'_1 \in \mathbf{U}(S_1)$ and $\exists S'_2 \in \mathbf{V}(S_1)$ such that

$$R_e(\vec{e}_t) = R_s(\vec{s}_t) \quad \text{and} \quad \vec{o}^e_t = \vec{o}^s_t \\ \Rightarrow \quad \forall E_1(\vec{e}_t, \vec{o}^e_t) \quad \exists S'_1(\vec{s}_t, \vec{o}^s_t) \qquad s.t. \\ R_e\big(E_1(\vec{e}_t, \vec{o}^e_t)\big) = R_s\big(S'_1(\vec{s}_t, \vec{o}^s_t)\big) \quad \forall \vec{s}_t, \vec{o}^e_t, \vec{o}^s_t, \vec{e}_t \tag{33}$$

and

$$\vec{R}_e(e_t) = \vec{R}_s(s_t) \\ \Rightarrow \quad \forall E_2(\vec{e}_t) \quad \exists S'_2(\vec{s}_t) \qquad s.t. \\ C_1\big(E_2(\vec{e}_t), c^e_t\big) = C_1\big(S'_2(\vec{s}_t), c^s_t\big) \quad \forall \vec{s}_t, \vec{e}_t \tag{34}$$

5 How to build minimal simulations for evolutionary robotics

In this section we outline how the minimal set of conditions for behavioural transference from simulation to reality that were derived in the last section can be practically applied to the building of minimal simulations for evolutionary robotics. This subject is dealt with in more detail in [4]. The basic idea behind the methodology is that reliably fit individuals in an evolutionary scenario are those that are able to consistently perform a certain behaviour, or set of behaviours, fitness trial after fitness trial after fitness trial. As we shall see, by randomly varying the simulation *between* trials, in ways put forward below, we can ensure that if a controller evolves to be reliably fit then it will be both base set exclusive, and base set robust. Provided the remaining members of the minimal set of conditions are fulfilled, all controllers that evolve to be reliably fit within the simulation will be able to reliably perform the behaviour in reality. Below we give a brief guide to building a minimal simulation, and relate it back to the minimal conditions for behavioural transference as we go. We then go on to examine ways of ensuring that the controllers that evolve to be reliably fit in such a simulation will transfer into reality.

5.1 The underlying model

In order to build a minimal simulation which we can use to evolve a particular robot behaviour, the first thing to do is to identify those features of the world that are to act as a basis for that behaviour. In other words, if the behaviour is defined by a particular $\mathbf{B_{env}}$ and B_{act}, we must decide on the base set of real world state variables that a suitable R_e can pick out in order that equation 29 is satisfied. Having done this, we can then build a simple model of how this base set of external state variables changes over time and interacts with the agent's controller. Such a model will involve the following components:

- An external state vector $\vec{s}_t$. This includes all the variables of the model that are involved in the generation of controller inputs and the processing of controller outputs.
- A reduction operator R_s. This does not need to be explicit, we just need to know which variables of the external state vector correspond to the base set, and that they satisfy the appropriate conditions.

- A function S_1. This is the function that governs how $\vec{s}_t$ changes over time and responds to controller output.

- A function S_2. This is the function that governs how controller input arises from the current $\vec{s}_t$.

The only restrictions on how we should go about choosing these components spring from the minimal set of conditions laid out in subsection 4.5. The model must be able to support the behaviour (equation 28), and we must be able to map all behaviourally relevant states of the world onto the model (equation 30). We must also have some idea of how *inaccurate* our model is so that we can come up with suitable sets of functions $\mathbf{U}(S_1)$ and $\mathbf{V}(S_2)$ that encompass accurate models of the way in which the base set of external state variables change over time (equation 33) and contribute to controller input (equations 34). If these conditions are fulfilled then controllers that are base set exclusive (equation 31) and base set robust (equation 32) with respect to $\mathbf{U}(S_1)$ and $\mathbf{V}(S_2)$ will be able to reliably perform the behaviour when downloaded onto the real robot.

5.2 How to ensure that reliably fit controllers are base set robust and base set exclusive

Let us imagine that we have built a simulation according to the methodology outlined above, and that we have therefore defined sets $\mathbf{U}(S_1)$ and $\mathbf{V}(S_2)$ with respect to which controllers that are base set exclusive and base set robust will reliably perform the behaviour in reality. How can we then ensure that controllers that evolve to be reliably fit in such a simulation will also be both base set exclusive and base set robust?

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 is able to perform the behaviour in question on all such trials. In order to ensure that the only controllers that can be reliably fit are those that are both base set robust and base set exclusive, therefore, all we have to do is to ensure that those that do not meet these conditions will be *unreliable*. This can be done by randomly varying the simulation from trial to trial in two ways: one that ensures reliably fit controllers will be base set robust, and one that ensures reliably fit controllers will be base set exclusive. Both will now be described.

In order to ensure that reliably fit controllers are base set robust we must ensure that they able to reliably perform the task in all $S' \rightleftarrows C$. This can be done by randomly varying the exact nature of the functions S_1 and S_2 from trial to trial. Controllers that can then perform the behaviour on all such trials must be robust to a wide variety of slightly different S_1 and S_2 functions. If we want to ensure that they are robust to all $S'_1 \in \mathbf{U}(S_1)$ and all $S'_2 \in \mathbf{V}(S_2)$, then at the start of each trial we must randomly pick a $S'_1 \in \mathbf{U}(S_1)$ and substitute it for S_1, randomly pick a $S'_2 \in \mathbf{V}(S_2)$ and substitute it for S_2, and then run the fitness trial as normal. Controllers that are reliably fit over all such trials must by definition be base set robust. In symbols, reliably fit controllers that evolve in such a simulation will be such that $\forall S'_1 \in \mathbf{U}(S_1)$ and $\forall S'_2 \in \mathbf{V}(S_1)$,

$$B_{act}(T_{R_s})\ \forall T_{R_s}$$

A controller that is base set exclusive is one whose behaviour depends exclusively on aspects of its input that derive from the values of the base set of external state variables. The method we put forward for ensuring that reliably fit controllers are base set exclusive is similar to that which ensures that they are base set robust. If a controller is reliably fit trial after trial, then those aspects of its input that contribute to its behaviour must in turn be reliable trial after trial. By ensuring that all those aspects of a controller's input that *do not* derive from the behaviourally defined base set of external variables are *unreliable*, from trial to trial, we can ensure that the controller's behaviour will not depend on these aspects if it is to be reliably fit. In other words, by ensuring that *the only* aspects of a controller's input that *are* reliable, trial after trial, derive from the behaviourally defined base set of external variables, we can ensure that reliably fit controllers will be base set exclusive.

Note that if a controller is base set exclusive, this does not mean that its behaviour necessarily depends on *all* aspects of the input that derive from the values of the base set of external variables. Just as we can prevent the behaviour of reliably fit controllers from depending on *non* base set aspects, so we can prevent the behaviour of reliably fit controllers from depending on certain aspects of the input that *do* derive from the base set. We might want to do this for several reasons. For instance, certain aspects of the controller's input might derive from the values of the base set in the simulation in ways that they cannot in reality, or our model of the way in which aspects of the controller's input derive from the values of the base set might be incomplete or inaccurate in certain respects. In fact, as we shall see, we only need to model *enough* of the way in which the real world base set contributes to controller inputs for the behaviour to be possible, all other aspects of the input can be made unreliable.

5.3 The potential power of minimal simulations

In the methodology outlined above it should be noted that there is no emphasis on the model being either particularly complex or particularly accurate: the main reasons why simulations can become overly computationally expensive and prohibitively time-consuming to

build. This is where the potential power of these techniques lie. A reliably fit controller that evolves in an inaccurate and simple minimal simulation 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. 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, one's choice of how to model the robot in its environment can be based on considerations of computational expense and ease of design rather than those of fidelity.

6 Experiments

In this section two sets of experiments are briefly outlined, both involving the evolution of non-trivial behaviours for real robots using minimal simulations. There is not enough room, unfortunately, to provide anything more than a flavour of how the theory of the last few sections can be employed in practice. For a fuller account, the reader is directed to [4]. In the first set of experiments controllers were evolved for the small mobile Khepera robot [6] that were able to solve a T-maze in response to a light signal. In the second set of experiments, controllers were evolved for the Sussex university gantry robot [3] that were able to visually distinguish a triangle from a square, and steer the robot towards it. We will look at each minimal simulation in turn.

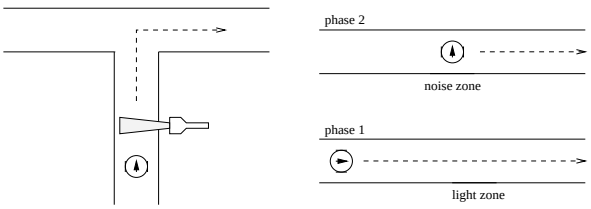

Figure 1: The task in simulation.

6.1 A minimal simulation of a Khepera robot

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 in the Evolutionary Robotics literature so far. The behaviour that was chosen is shown diagrammatically on the left of figure 1. 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. The minimal simulation used in the experiments was designed with low computational overheads firmly in mind. To give some idea of its simplicity, it contained two look-up-tables, one containing 72 values and one containing 80, and about 300 lines of commented C code that employed nothing more mathematically complicated than floating point arithmetic.

The two phases of the T-maze simulation are shown in figure 1. The base set of external state variables that it models consists simply of the relative positions of the robot, the light signal and the walls of the T-maze environment. In the first phase, the virtual robot travels down a simple corridor where it receives a light signal from either one side or the other. After it has travelled a predetermined distance, it is 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 has to chose whether to turn left or right, depending on which side the light signal was on, in order to gain maximum fitness points.

Now although this twin corridor set-up varied significantly from a T-maze, evolving control architectures could interact with it *as if it were* a T-maze (equation 28). In particular, for every position of the Khepera within the real world there was a corresponding position within the simulation (equation 30). The model of the way in which the Khepera moves in response to motor signals, and the model of the way in which the infrared sensors and ambient light sensors return values as a function of position were all extremely simple. They were, however, near enough to reality that suitable sets of functions $\mathbf{U}(S_1)$ and $\mathbf{V}(S_2)$ could be generated just by varying their parameters. Reliably fit control architectures were forced to be base set robust, therefore, by randomly varying these parameters from trial to trial. Where those aspects of the controller's input due to the position of the virtual robot within the simulation varied significantly from those aspects of the controller's input due to the position of the real robot within the T-maze (i.e. around the area marked 'noise zone' in figure 1), these aspects were made completely unreliable within the simulation so that reliably fit controllers were forced to employ strategies that in no way depended upon them.

Using this minimal simulation, reliably fit recurrent neural network controllers consistently evolved within around 1000 generations, the simulated equivalent of over 17 months of continuous real-world evolution. This

took around 4 hours to run as a single user on a SPARC Ultra. In order to see whether it would successfully transfer across the reality gap, one of the first reliably fit networks to evolve was downloaded onto a Khepera robot, and its ability to perform the task in the real world was tested. Sixty different trials were performed one after another, twenty in each of three different widths of corridors (11cm, 18cm and 23cm), with the light on the left for ten trials and the light on the right for the other ten. On all sixty occasions the Khepera performed the task satisfactorily and efficiently, navigating to the end of the correct arm of the T-maze without colliding with the walls.

6.2 A minimal simulation of the gantry robot

In [3], 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 for the Sussex university gantry robot. This robot, although suspended from a gantry, is best thought of as a small wheeled robot with a camera placed on top looking horizontally forwards. Starting from several different positions and orientations, evolving individuals steered the robot, using visual input, around an all-black rectangular arena with a white triangle and a white square stuck upon one of the long walls. Maximum fitness points were awarded to those controllers that could visually discriminate the triangle from the square and steer the robot towards it. In the experiments reported here, both neural network controllers and their visual morphologies were again evolved to perform the same triangle-square discrimination task. The only difference being that they were evolved to perform it under extreme real world noise provided by a set of lights, each turning on and off at different frequencies, known at Sussex as the 'disco lights'.

The base set of external state variables that were modelled consisted of the relative positions of the robot, the walls of the arena, the triangle and the square. The geometry of the simulation was based on that of reality in that there was a virtual rectangular arena with a virtual robot moving around within it. The simulation was therefore able to support the behaviour (equation 28), and every behaviourally relevant real-world state could be realised within it (equation 30). The model of the way in which the base set of external state variables variables changed over time and responded to motor signals was again simple and look up table based. The location in the arena that each individual pixel projected onto was calculated using simple trigonometry (with look up tables for sin, cos and tan),as a function of the position of the robot within its environment. Having done this, the way in which pixel values were assigned is outlined below. Both models were again near enough to the real situation that suitable sets of functions $\mathbf{U}(S_1)$ and $\mathbf{V}(S_2)$ could be generated just by varying their parameters. Reliably fit controllers were therefore forced to be base set robust by randomly varying these parameters from trial to trial.

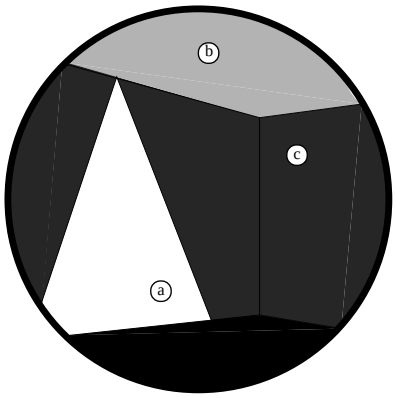

Figure 2: A typical image returned by the camera of the gantry robot. The robot is facing the corner of the arena and the triangle can be seen on the left. The white circles labelled a, b and c are examples of pixels that project onto the triangle, ceiling and wall respectively. Pixel a will return a 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.

Under the 'disco lights' suspended above the gantry, the values returned by pixels of the camera-image varied widely both with respect to time, and with respect to the direction of the camera. Even if we knew the exact location within the arena that a particular pixel projected onto, therefore, there was not that much we could say about exactly what the value of that pixel might be. However, there were a few general things that held true except in exceptional circumstances: if a pixel projected onto a wall but not onto a shape then it returned a value within the range 0 to 13, if a pixel projected onto either the triangle or the square then it returned a value between 14 and 15, and if a pixel projected onto either the floor or the ceiling of the arena it returned a value between 0 and 15 (see figure 2). Since these facts about the intervals that pixel values fell into under full 'disco lighting' were almost always the case, and since they were enough to distinguish the white triangle and square from the black walls of the arena, they were the only aspects of the controller's input (which derived from the base set of external state variables) that we needed to model. All other aspects of the input, including *the way in which* pixel values were assigned within these intervals, were therefore varied from trial to trial in a way that made them totally unreliable. In this way, reliably fit individuals were forced to depend solely on the fact that pixel values that projected onto certain surfaces fell

within certain intervals, and not on how or where they fell within these intervals. They were therefore forced to be base set exclusive.

Controllers (and their visual morphologies) that were reliably fit within the simulation consistently evolved within 6000 generations, which is the simulated equivalent of over 3 years worth of continuous real-world evolution. This took around 12 hours to run as a single user on a SPARC Ultra. One of the most efficient networks was chosen for testing: when placed in one of the four starting positions in the arena, the network initially caused the robot to turn in a tight circle clockwise. If the square came into the view of the camera, the rotational speed of the robot would actually increase until the square was out of view. When the triangle came into view, the robot would then 'lock on' and move directly towards it, adjusting its course as it went.

In order to see whether it would cross the reality gap, the network was downloaded onto the gantry, and tested continuously[3] 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 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 small radius extending out from the centre of the triangle. In total the robot achieved the task 195 times out of 200: 98 with the triangle on the right and 97 with the triangle on the left. All tests were performed under full 'disco lighting'.

7 Conclusions

In the first two sections of this paper a theoretical and notational framework was developed that allowed us to formally state the circumstances under which an agent performs a behaviour within an environment. This framework was then used to prove how an agent's reliable behaviour within one agent-environment system could imply its reliable behaviour within another providing that certain conditions were fulfilled by the nature of the controller and the relationship between the two systems. This led to the derivation of a minimal set of conditions that must be fulfilled by the simulation, the controller, and the reliable behaviour of the controller within the simulation, if we are to guarantee that the controller will also be able to reliably perform the behaviour in reality. The next section outlined how this minimal set of conditions for behavioural transfer could be practically applied to the building of minimal simulations for evolutionary robotics. Techniques were then put forward for ensuring that controllers which evolved to be reliably fit within such simulations would continue to be reliably fit when downloaded into reality. Finally, two sets of experiments were sketched in which minimal simulations were used to evolve controllers for non-trivial real world tasks. In both sets of experiments, controllers evolved in a matter of hours that were able to perform the behaviours reliably and robustly when downloaded onto the real robots.

[3] In practice, because of the propensity of the mechanics of the gantry robot to seize 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.

Acknowledgements

Special thanks to Joe Faith for help with getting the notation right, and to Phil Husbands and Jon Bird for proof reading. Thanks also to the school of COGS for the bursary that allows me to undertake this work.

References

[1] R.D. Beer. A dynamical systems perspective on agent environment interaction. *Artificial Intelligence*, 72(1-2):173–215, 1995.

[2] R.D. Beer and J.C. Gallagher. Evolving dynamic neural networks for adaptive behavior. *Adaptive Behavior*, 1:91–122, 1992.

[3] I. Harvey, P. Husbands, and D. Cliff. Seeing the light: Artificial evolution, real vision. In D. Cliff, P. Husbands, J.A. Meyer, and S. Wilson, editors, *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, volume 3. MIT Press/Bradford Books, 1994.

[4] N. Jakobi. Evolutionary robotics and the radical envelope of noise hypothesis. Cognitive Science Research Paper CSRP457, University of Sussex, 1997.

[5] N. Jakobi, P. Husbands, and I. Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In F. Moran, A. Moreno, J.J. Merelo, and P. Chacon, editors, *Advances in Artificial Life: Proc. 3rd European Conference on Artificial Life*. Springer-Verlag, 1995.

[6] K-Team. Khepera users manual. EPFL,Lausanne, June 1993.

[7] M.J. Mataric and D. Cliff. Challenges in evolving controllers for physical robots. *Robot and Autonomous Systems*, 19(1):67–83, 1996.

[8] Olivier Michel. An artificial life approach for the synthesis of autonomous agents. In J.M. Alliot,

E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, *Proceedings of the European Conference on Artificial Evolution*. Springer-Verlag, 1995.

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

[10] T. Smithers. What the dynamics of adaptive behaviour and cognition might look like in agent-environment interaction systems. In T. Smithers and A. Moreno, editors, *3rd International Workshop on Artificial Life and Artificial Intelligence, The Role of Dynamics and Representation in Adaptive Behaviour and Cognition*, San Sebastian, Spain, 1994.