CSRP 497
Running Across the Reality Gap: Octopod Locomotion Evolved in a Minimal Simulation*

Nick Jakobi

School of Cognitive and Computing Sciences,
University of Sussex, Brighton, BN1 9QH, UK
nickja@cogs.susx.ac.uk

Abstract. This paper describes experiments in which neural network control architectures were evolved in minimal simulation for an octopod robot. The robot is around 30cm long and has 4 infra red sensors that point ahead and to the side, various bumpers and whiskers, and ten ambient light sensors positioned strategically around the body. Each of the robot's eight legs is controlled by two servo motors, one for movement in the horizontal plane, and one for movement in the vertical plane, which means that the robots motors have a total of sixteen degrees of freedom. The aim of the experiments was to evolve neural network control architectures that would allow the robot to wander around its environment avoiding objects using its infra-red sensors and backing away from objects that it hits with its bumpers. This is a hard behaviour to evolve when one considers that in order to achieve any sort of coherent movement the controller has to control not just one or two motors in a coordinated fashion but sixteen. Moreover it is an extremely difficult set-up to simulate using traditional techniques since the physical outcome of sixteen motor movements is rarely predictable in all but the simplest cases. The evolution of this behaviour in a minimal simulation, with perfect transference to reality, therefore, provides essential evidence that complex motor behaviours can be evolved in simulations built according to the theory and methodology of minimal simulations.

1 Introduction

Evolutionary Robotics is not magic and as several authors have pointed out [2, 6, 12, 14], there are many big questions that need answers if Evolutionary Robotics is to progress beyond the proof of concept stage. One of the most urgent of these (in that if it is not answered, Evolutionary Robotics is not going to progress very far at all) concerns how evolving controllers should best be evaluated. If they are tested using real robots in the real world, then this has to be done in real time, and the evolution of complex behaviours will take a prohibitively long time. If controllers are tested using simulations then the amount of modelling necessary to ensure that evolved controllers work on the real robot may mean that the simulation is so complex to design and so computationally expensive that all potential speed advantages over real-world evaluation are lost. How then should controllers be evaluated when testing in both simulation and reality seems fraught with insurmountable problems?

The author's recent thesis [9] offers an answer to this question. It does this by presenting new ways of thinking about and building simulations for the evaluation of evolving robot controllers. These *minimal simulations* run extremely fast and are trivially easy to build when compared to more conventional types of real-world simulation, yet they are still capable of evolving controllers for real robots. Thus the many advantages of using simulations are preserved while most of the major disadvantages are avoided. The aim of this paper is to show how complex motor behaviours can be evolved in minimal

* A version of this paper appears in P. Husbands and J.-A. Meyer (Eds), *Evolutionary Robotics*, Springer Verlag LNCS vol. 1468, 39–58, 1998.

simulation. Space limitations do not allow a full explication of the theory and methodology of minimal simulations. However, the next section gives a broad brush sketch of the main ideas, which, together with the details of the minimal simulation used for the experiments reported in this paper, should give the reader a good idea of how to build and use a minimal simulation.

Fig. 1. The Octopod robot.

This paper describes experiments in which neural network control architectures were evolved for an octopod robot. The robot, shown in figure 1 is around 30cm long and has 4 infra red sensors that point ahead and to the side, various bumpers and whiskers, and ten ambient light sensors positioned strategically around the body. Each of the robot's eight legs is controlled by two servo motors, one for movement in the horizontal plane, and one for movement in the vertical plane, which means that the robots motors have a total of sixteen degrees of freedom.

The aim of the experiments was to evolve neural network control architectures that would allow the robot to wander around its environment avoiding objects using its infra-red sensors and backing away from objects that it hits with its bumpers. This is a hard behaviour to evolve when one considers that in order to achieve any sort of coherent movement the controller has to control not just one or two motors in a coordinated fashion but sixteen. Moreover it is an extremely difficult set-up to simulate using traditional techniques since the physical outcome of sixteen motor movements is rarely predictable in all but the simplest cases. The evolution of this behaviour in a minimal simulation, therefore, provides essential evidence that complex motor behaviours can be evolved in simulations built according to the theory and methodology put forward in [9, 8, 7]. See [3, 4, 11] for related work on evolving locomotion controllers for walking robots and for abstract computer models of insects.

The minimal simulation used to evolve controllers for the octopod is described in Section 3. The rest of the evolutionary machinery, including the neural networks, the encoding scheme, the genetic algorithm and genetic operators is described in section 4. Experimental results are put forward in section 5 and finally, in section 6, some comments are offered on the paper as a whole. But first, a brief description of the idea of minimal simulations.

2 Minimal simulations

The artificial evolution of controllers 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 [15, 14]) can be done in only one of two ways: controllers 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 [12] point out, the evaluation of controllers on real robots must be done in real time, and this probably makes the entire evolutionary process prohibitively slow. 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, for instance. This begs many technological questions to do with power supplies, wear and tear, automatic fitness evaluations ans so on.

As has been shown by several experimenters [10, 1, 13], it *is* possible to evolve controllers in simulation for a real robot. Now that this is no longer in doubt the question becomes one of whether the technique will scale up. In [12] (and similar points were made earlier in [6, 2, 5]), 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 and methodological framework that enables the easy and cheap construction of fast-running simulators for evolving real-world robot behaviours. This is what is provided in [9, 8, 7] where the general framework is developed and a wide range of examples are given in which evolved behaviours successfully crossed the reality gap (the evolved controllers performed perfectly on the real robots). The basic idea of a minimal simulation is to model only those robot-environment interactions that are necessary to underpin a desired behaviour. Everything else is made unreliable by careful use of randomness. In this way an evolved controller is forced to use the minimal set of interactions picked out by the simulation designer. This set of interactions is modelled as simply (i.e. computationally cheaply) as possible by using an envelope of noise to mask the inaccuracies of the modelling. Again, by careful use of randomness, controllers evolve that are robust to these inaccuracies and hence will cross the reality gap.

The whole methodology can be summarised in the following step-by-step guide to building a minimal simulation:

1. **Precisely define the behaviour.** Start by making a precise definition of the behaviour to be evolved. This should include both a description of the task to be performed by the robot(s) and the range of environmental conditions it is to be performed under.
2. **Identify the real-world base set.** Distinguish between those real-world features and processes that are relevant to the performance of the behaviour and those that are not. Those that are relevant constitute the base set. If possible, identify the way in which the members of the base set interact with each other and react to motor signals during the performance of the behaviour.
3. **Build a model of the way in which the members of the base set interact with each other and react to controller output (when the robot is performing the behaviour).**

Make a model of the real-world base set of features and processes. The dynamics of this model need copy those of the real world only during the performance of successful behaviour. The dynamics when the behaviour is not being performed may often be shaped to smooth the fitness landscape, thus facilitating the evolution of successful controllers.

4. **Build a model of (enough of) the way in which the members of the base set affect controller input (when the robot is performing the behaviour).** Identify and model processes by which members of the real-world base set give rise to aspects of controller input. Just as with the model of the base set, these modelled processes need only copy their real-world counterparts when the behaviour is being performed. Make sure that the input aspects that these processes give rise to in the minimal simulation are sufficient to underly successful behaviour. Note that there are often several ways in which sufficient input processes may be identified and modelled, and the exact choice affects the possible strategies that evolving controllers may employ.
5. **Design a suitable fitness test.** Design a suitable fitness test that only awards maximum fitness points to those controllers that reliably perform the behaviour. In particular, each evaluation must involve a sufficient number of trials so that, with the right amount of trial to trial variation (see below), we can be confident that controllers which achieve high fitness in all of them are reliably fit, base set exclusive and base set robust.
6. **Ensure that evolving controllers are base set exclusive.** Make a distinction between the implementation aspects (those features of the simulation that are there for coherence etc but which we do not want to underpin the desired behaviour) of controllers' input signals and the base set aspects. Those implementation aspects that are present during the performance of the behaviour must be randomly varied from trial to trial so that evolving controllers that depend upon them are unreliable. In particular, enough variation must be included to ensure that evolving controllers can not, in practice, be reliably fit unless they are base set exclusive i.e. they actively ignore each implementation aspect and depend exclusively on the base set aspects of their input to perform the behaviour.
7. **Ensure that evolving controllers are base set robust.** Every base set aspect of the simulation must be randomly varied from trial to trial so that reliably fit controllers are forced to be base set robust. The extent of this random variation must be large enough that controllers which evolve to be reliably fit are also able to cope with the inevitable differences between the base set aspects of the simulation and their real-world counterparts. Care should be taken that this variation is not so large that reliably fit controllers fail to evolve at all.

3 The Octopod minimal simulation

According to received wisdom, simulating something as complex from an actuator point of view as an eight-legged robot is hard. The problems arise from the fact that sixteen motors all moving at the same time and interacting with each other in the real world rarely induce movement in the robot that is easy to model and often produce completely unpredictable movement that is best looked at as stochastic. What happens when two legs clash, for instance? Or when the belly of the robot is on the ground but the legs attempt to push the robot forwards? Or when 4 of the legs attempt to push the robot forwards and 4 of the legs attempt to push the robot backwards? Clearly any simulation that sets out to model *all* of the dynamics of the system will involve vast quantities of pain-staking empirical measurement and research into friction-coefficients, the power of each motor, the range of possible movement of the robot and so on. If the only simulation in which we could evolve autonomous walking behaviour for the real robot was of this type then the simulation would be so complicated that it might indeed be simpler to evolve controllers on the real thing.

Happily we do not need to come close to modelling all of the possible dynamics of the robot in order to build a satisfactory minimal simulation. The key is to realise that those portions of the possible dynamics of an octopod robot which are difficult and complicated to model (the vast majority) are precisely those that are *not* involved in successful walking behaviour. When the octopod robot walks

around its environment in an acceptable manner, its legs do *not* clash and its belly does *not* drag along the ground and its legs do *not* pull in different directions. The minimal simulation described below takes full advantage of this fact. The dynamics of the simulated robot match the dynamics of the real robot only when the controller is inducing acceptable, successful walking and obstacle-avoiding behaviour. If a controller does anything else *but* acceptable, successful walking and obstacle-avoiding behaviour then the simulation falls woefully short of modelling what would actually happen in the real world. Since a controller that performs the behaviour will never take the robot into this region of the dynamics, we do not need to model it. The headings for the step-by-step guide to minimal simulations given in section 2 will be followed in describing the octopod simulation in detail.

Precisely define the behaviour.

The aim of the experiments was to evolve octopod-controllers that could walk around the environment, turning away from objects that fell within range of the IR sensors and backing away from objects that touched the front bumpers and whiskers. At the very least, this requires that controllers are able to perform 4 sub-behaviours, each relevant to a particular sensory scenario:

- If an object falls within range of the left-hand IR sensors then the robot must turn on the spot to the right.
- If an object falls within range of the right-hand IR sensors then the robot must turn on the spot to the left.
- If an object hits the front bumpers or whiskers then the robot must walk backwards as fast as possible.
- In the absence of any objects falling within infra-red range or touching the robot's front bumpers or whiskers, the robot must walk forwards in a straight line as fast as possible.

In a cluttered environment there are occasions in which other sensory combinations may occur e.g. objects may fall within range of the left and right IR sensors at the same time. However, these occasions are rare enough in simple environments to grant that controllers which are able to perform each of these 4 simple sub-behaviours are also capable of wandering around their environment satisfactorily without bumping into anything or becoming stuck.

One reason for making this behavioural reduction is that constructing a fitness test that specifically checks for each of the 4 sub-behaviours, one after the other, is actually much easier than constructing a fitness test that checks directly for the more complex global behaviour. We do not need to simulate, for example, the way in which the robot's position within a complex environment gives rise to sensor values. Instead we may test directly for each of the 4 sub-behaviours in turn by clamping the sensor values to fit each of the 4 sensory scenarios and observing the movement of the robot in response. The fitness function was therefore divided into 4 phases: each testing for one of the 4 behaviours outlined above. The order in which each of the 4 phases occurred was random and evolving neural network controllers were not reset in between. This ensured that reliably fit controllers would be able to perform each of the 4 sub-behaviours independently.

Identify the real-world base set

Whether or not the robot satisfactorily performs each of the 4 sub-behaviours is a function of the movement of the robot body in each of the 4 different sensory scenarios. The members of the base set, therefore, are those features of the world that make up the causal pathway from controller output to

how the body as a whole moves in response. These include the way in which controller output affects how the legs move, and the way in which the movement of the legs affects the movement of the body as a whole.

Build a model of the way in which the members of the base set interact with each other and react to controller output (when the robot is performing the behaviour).

The overall movement of the robot was described by two variables: one for the speed of the left-hand side of the robot and one for the speed of the right-hand side of the robot. Thus if both sides of the robot moved straight ahead at the same speed then the overall movement of the robot was deemed to be straight ahead, if they moved in different directions but with equal velocity then the robot was deemed to be turning on the spot, and if both sides moved backwards at the same speed then the overall movement of the robot was deemed to be straight backwards.

To model the way in which the robot as a whole moved in response to controller output, therefore, necessitated a model of the way in which each leg responded to controller output, and the way in which the movement of each leg contributed to the overall movement of each *side* of the robot. However, because of the arguments put forwards in [9], it was not necessary to accurately model the way in which *every* motor signal could affect the movement of the robot as a whole, but only those motor signals involved in satisfactory walking forwards, backwards and turning on the spot. The dynamics of the model, therefore, matched those of reality only for those controllers that prevented the body from touching the ground, moved all the legs supporting the robot on each side in the same direction (either all forwards or all backwards depending on whether the robot was supposed to be walking forwards, backwards or turning on the spot), and kept those legs that were not touching the ground as high in the air as possible.

The motor signals to the servo-motors controlling the legs of the octopod robot specify absolute angular positions (relative to the body) that the servo-motors are required to move the legs to. Thus when a new signal is sent to the servo-motor controlling the horizontal or vertical angle of a particular leg, it will move as fast as possible to the new location. In the absence of any new signal, the leg will remain rigid. This process was modelled in the simulation by calculating, on every iteration, horizontal and vertical angular displacements for each leg based on the differences between the angular positions specified by the motor signals and the actual angular positions of the simulated legs. The maximum possible angular speed of each leg was measured very roughly and set in the simulation to be 2π radians per second. Using the horizontal and vertical angles of each leg, a simple look up table provided the approximate position, relative to the robot, that each leg projected onto the ground, and the 4 legs in the lowest positions were assigned as the supporting legs. A simple calculation was then made to see whether the robot's centre of gravity was contained within the polygon subtended by the floor-contact positions of these 4 legs, in which case the robot was deemed to be stable. If it was not, then the robot was deemed to be unstable. Also the average height of these 4 legs relative to the robot body was calculated. If they were low enough then the robot was deemed to be standing, otherwise it was deemed to be dragging its belly on the ground.

Figure 2 shows diagrammatically how the speeds of the left and right-hand sides of the robot, and thus the overall movement of the robot, were calculated from the controller's motor signals. On each iteration, the contribution each leg made to the forwards or backwards movement of its side of the robot was worked out according to a simple calculation. The distance moved by the leg (either forwards or backwards) was multiplied by a figure between 0 and 1 that was inversely proportional to how high in the air the leg was. Thus the higher in the air a leg was, the smaller the contribution its horizontal movement made to the total movement of its side of the robot. The nearer to the ground the leg was, the larger the contribution its horizontal movement made to the total movement of its side of the robot. The contributions that each leg makes were then added up to arrive at a figure for the total movement

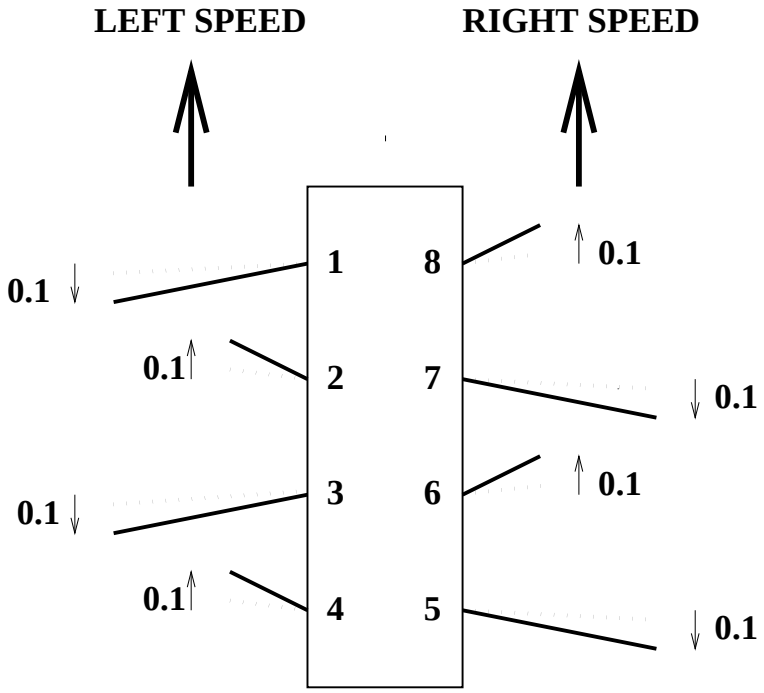

Fig. 2. This figure shows diagrammatically how the speeds of the left-hand and right-hand sides of the robot were calculated from the vertical and horizontal positions of the eight legs. For explanatory purposes the length of each leg in the diagram is inversely proportional to its height above the ground so that the long legs are 0.8 as low as they can go and the short legs are 0.2 as low as they can go. Adding up the contributions that each leg makes to the speed of its side we see that the speeds of both the left and the right hand side of the robot work out at $0.1 \times 0.8 - 0.1 \times 0.2 + 0.1 \times 0.8 - 0.1 \times 0.2 = 0.12$ forwards

(either forwards or backwards) of that side of the robot. If both the left and the right side of the robot moved forwards then the robot was deemed to have moved forwards, if both sides moved backwards then the robot was deemed to have moved backwards, if each side moved in different directions then the robot was deemed to be turning on the spot.

Now although this simple model seems to bear no relationship to reality (how can a leg that is in the air contribute to the speed of its side of the robot?), a controller that made maximum use of the dynamics of the model to move the robot around as fast as possible would keep all of the legs that were moving in the wrong direction at any one time as high in the air as possible and all the legs that were moving in the appropriate direction as firmly on the ground as possible. Since penalty terms for both instability and belly-dragging were included in the fitness function (see below), maximally fit controllers remained stable and stood upright at all times, moving all the legs that were supporting the robot on each side in the same direction (either all forwards or all backwards depending on whether the robot was walking forwards, backwards or turning on the spot) and keeping those legs that were not supporting the robot as high in the air as possible.

Build a model of (enough of) the way in which the members of the base set affect controller input (when the robot is performing the behaviour).

The sensor model employed was so simple as to be almost non-existent. The sensors were divided into three groups: the front left and back left IR sensors forming one group, the front right and back

right IR sensors forming another group, and the front whiskers and bumpers forming another. In the phase of each fitness test in which there were no objects within sensor range, all sensors were set to background levels for the duration of the phase: 0 for the bumpers and whiskers and 255 for the IR sensors. In the phases during which an object fell within IR range on either the left or right-hand side of the robot, the IR sensor on the appropriate side was set to high (200) for the duration of the phase. In the phase during which an object hit the touch sensors, the front whiskers and bumper were set to high (1), *but only for the first second of the phase*. This simple sensor model provided evolving controllers with enough information about the world to perform the behaviour satisfactorily.

Design a suitable fitness Test

As explained above, each fitness evaluation was divided into 4 phases: one for each of the 4 sensory scenarios. Each of these phases lasted five simulated seconds. At the end of every iteration of the simulation, the fitness of the controller being tested was incremented by a value δ derived from the overall movement of the robot. How this value was calculated depended on the sensory scenario the robot was in at the time:

- If there were no objects within sensor range then δ was the speed of the left-hand side of the robot plus the speed of the right-hand side of the robot.
- If there was an object within infra-red sensor range on the right-hand side of the robot then δ was the speed of the right-hand side of the robot *minus* the speed of the left-hand side of the robot.
- If there was an object within infra-red sensor range on the left-hand side of the robot then δ was the speed of the left-hand side of the robot *minus* the speed of the right-hand side of the robot.
- If an object hit the bumpers then δ (for the duration of this phase of the fitness evaluation) was *minus* the speed of the left-hand side of the robot *minus* the speed of the right-hand side of the robot.

Also on each iteration, if the robot was deemed to be unstable then a small penalty was subtracted from the fitness, and if the robot was deemed to be touching the ground with its belly then a small penalty was subtracted from the fitness.

Ensure that evolving controllers are base set robust and base set exclusive

The fitness test described above was carefully designed so that controllers that evolved to be reliably fit would use only those portions of the simulation dynamics that corresponded closely to the dynamics of the real robot. In fact, these dynamics turned out to be close enough that there was no need to vary the simulation at all in order to ensure that evolved controllers were base set robust. Any differences between simulation and reality were easily accommodated by slop in the definition of satisfactory walking and obstacle-avoiding behaviour. Thus walking behaviour on the real robot might be a little jerkier or quicker than in the simulation, but it was still perfectly adequate walking behaviour.

Likewise, nothing extra was added to the simulation in order to ensure that evolving controllers were base set exclusive. This was for the simple reason that the model of the way in which sensor values arose from the base set was so simple that there was nothing else in the simulation that evolving controllers could come to rely upon.

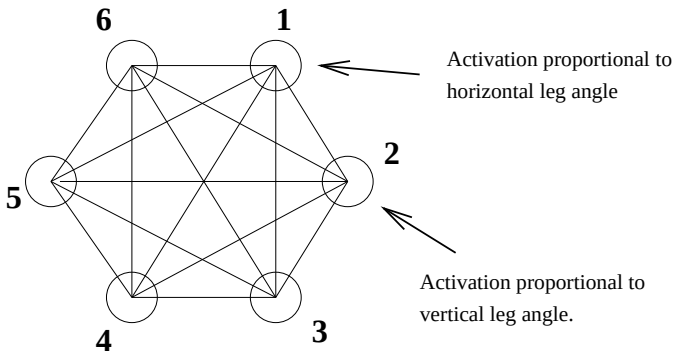

Fig. 3. Each leg controller consisted of six fully connected neurons. The activity of neuron 1 and neuron 2 controlled the horizontal and vertical leg angles respectively.

4 The evolutionary machinery

In this section we describe the evolutionary machinery that, together with the minimal simulation described above, was responsible for evolving neural network control architectures that could perform the behaviour satisfactorily in reality.

Neural networks

While network parameters (connection weights, time constants, thresholds and so on) were under evolutionary control in the experiments described in this section, the overall shape of the network architecture was fixed to be the same for every member of the population. The repetitive movements characteristic of multi-legged walking behaviours were produced by a main oscillatory network of 8 coupled sub-networks, each responsible for the direct control of a single leg. The properties of this oscillatory network were then modulated by the output from three sensory neurons (one each for left and right infra red and one for the bumpers) and one permanently saturated bias neuron to produce the different movement patterns for walking forwards, backwards and turning. This architecture is very similar to, and was based upon, that used by Gallagher and Beer in [1]. The components of this architecture will now be explained in detail.

Figure 3 shows one of the basic sub-networks responsible for the control of each leg. All eight sub-networks were identical in that only one set of sub-network parameters (threshold constants, connection weights and so on) was encoded on the genome and repeated eight times. These sub-networks consisted of six fully interconnected neurons, numbered 1 to 6 in the diagram, of the same type as those used by Gallagher and Beer in [1]. At each iteration, the input activity A_j of each of the $j = 1 \ \ to \ \ 6$ neurons in each of the 8 sub-networks was calculated according to equation 1

$$\tau_j \dot{A_j} = -A_j + \sum w_{ij} O_i + I_j \tag{1}$$

where τ_j was a time constant that affected the rate and extent to which the jth neuron responded to input, O_i was the output from the i_{th} neuron, w_{ij} was the weight on the connection from the i_{th} neuron to the j_{th} neuron, and I_j was any external input to the j_{th} neuron from outside the network. Once the input activity of each neuron had been calculated, the output O_j of each of the $j = 1 \ \ to \ \ 6$ neurons in

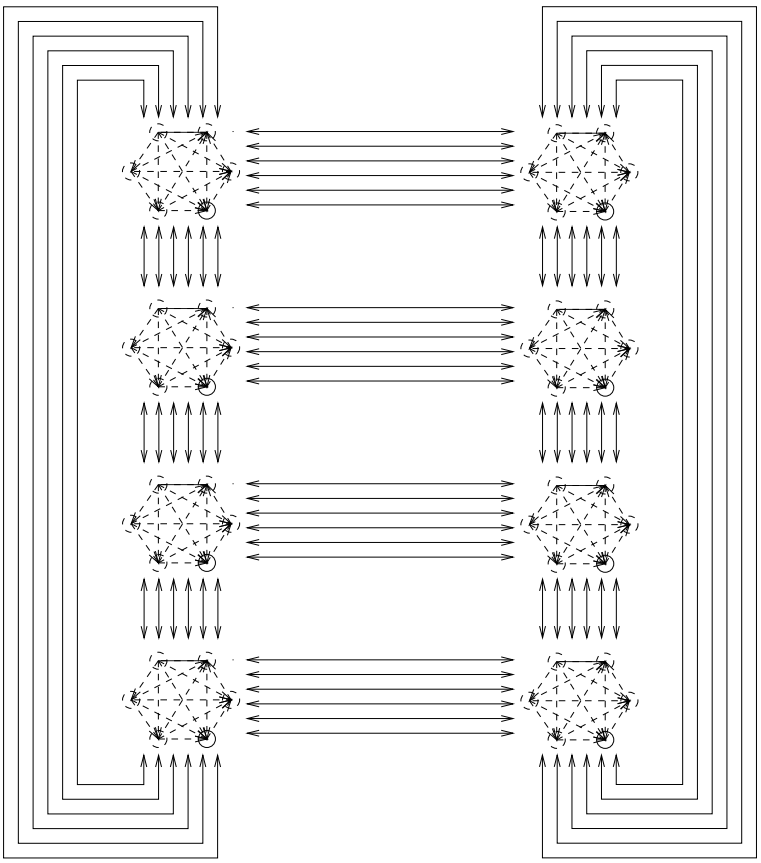

Fig. 4. This diagram shows how each leg-controller sub-network was coupled to the sub-network opposite it on the body and to the sub-network either side of it with wraparound.

each of the 8 sub-networks was calculated from the input activity A_j according to the sigmoid function of equation 2

$$O_j = (1 - e^{(t_j - A_j)})^{-1} \tag{2}$$

where t_j was a threshold constant associated with the j_{th} neuron. The range of possible values of each of these genetically specified constants is listed below.

The output of neuron 1 in each sub-network was responsible for the signal to the servo-motor controlling the horizontal motion of the leg in question, and the output of neuron 2 was responsible for the signal to the servo-motor that controlled the vertical angle of the leg (see figure 3). For neuron 1, an output of 0 mapped onto a signal to the horizontal servo motor to point as far backwards as it could go, and an output of 1.0 mapped onto a signal to the the servo motor to point the leg as far forward as it could go. For neuron 2, outputs of 1 and 0 mapped onto signals to the vertical servo motor to position the leg in the fully up and down positions respectively.

Each sub-network was coupled to the sub-network directly opposite it and to the network on either side of it (with wraparound) as in figure 4. Each sub-network to sub-network coupling involved six

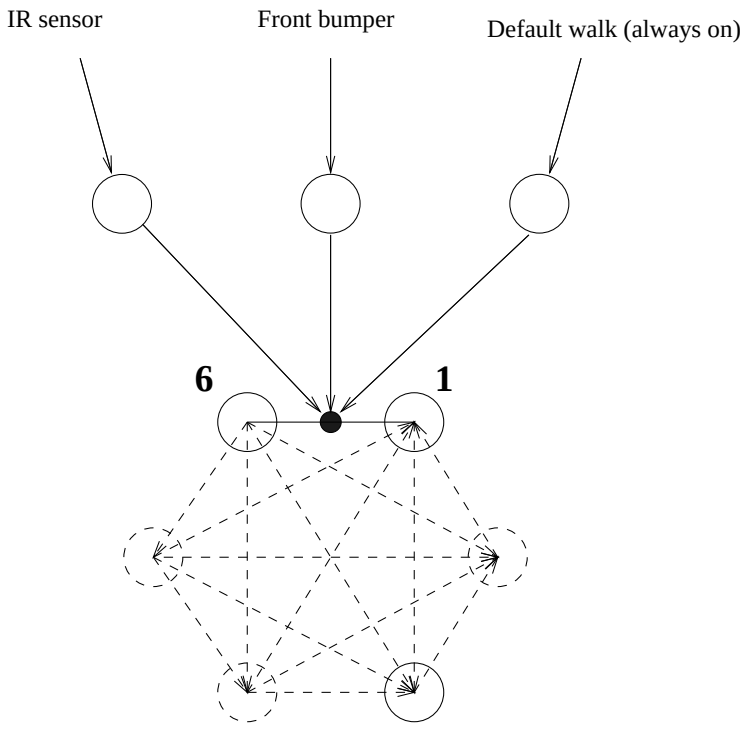

Fig. 5. This diagram shows how each connection between leg-controller neurons contains a synapse 'gate' that can be turned on or off by sensor neurons and the bias neuron. For the sake of diagrammatic simplicity only one connection is shown, whereas in reality every connection in every leg controller sub-network ($36 \times 8 = 288$ connections in total) contains a synapse that can be modulated in this way.

symmetrical connections: from neuron 1 in one network to neuron 1 in the other, from neuron 2 to neuron 2, neuron 3 to neuron 3 and so on. All 4 cross-body couplings were identical in the sense that only six connection strengths were encoded on the genome and this set of six was repeated 4 times. All 8 along-body couplings were identical in the same way.

Figure 5 shows an example of how the connections between the neurons that made up the leg-controller sub-networks could be modulated by the sensor neurons and the bias neuron. Each connection between leg-controller neurons can be thought of as having had a synapse half way down its length that acted as a gate: open and the connection was unaffected, closed and the connection was switched off, effectively reducing the weight on the connection to zero. The synapse itself received input from sensor neurons and the bias neuron by way of weighted connections. If the total input to the synapse was greater than zero then the synapse gate was open and the connection between the leg-controller neurons was unaffected. If the total input to the synapse was less than zero then the synapse gate was closed and the weight on the connection between the leg-controller neurons dropped to zero.

Figure 6 shows how the three sensor neurons and the bias neuron were connected up to the synapses of the leg-controller sub-networks. Each of the thick black arrows represents 36 connections, one for each of the 36 synapses of a leg-controller sub-network. In total, three sets of 36 connection weights were encoded on the genome: one set for the infra-red sensor neurons, one for the bumper sensor neuron and one for the bias neuron. Thus each of the two infra-red sensor neurons were connected to the synapses of the leg-controller sub-networks on the appropriate side by way of four identical sets of 36 connections (both sets of four were also identical to each other), and both the bumper sensor neuron

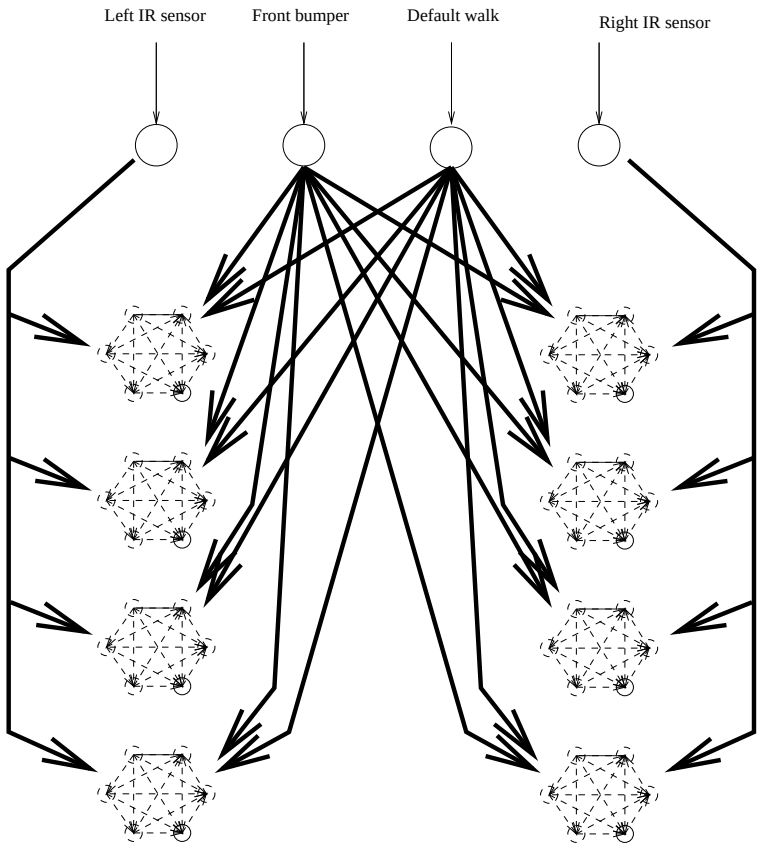

Fig. 6. This diagram shows how the sensory neurons and the always-on bias neuron were connected to the synapses of the leg controller sub-networks. Each of the thick black arrows represents 36 connections, one to each synapse in the leg controller. The cross-body and along-body couplings between leg-controller sub-networks have not been shown in this diagram.

and the bias neuron were connected up to all eight leg-controller sub-networks by way of eight identical sets of connections each.

A weighted input connection was associated with each of the three sensor neurons and the bias neuron. The signals from the infra-red sensors and bumper sensors that fed into these connections were normalised to lie within the range 0 to 1. In the case of the bias neuron, the signal that fed into its weighted input connection was permanently set at 1.

The network was updated iteratively using time-slicing techniques at a rate of 16 updates per second (or the simulated equivalent of a second). Also, in order to reduce computational overheads, a 200 place look-up-table was provided for the sigmoid function in place of the standard C-library maths functions.

Encoding scheme

Since the layout of the neural network architecture was fixed and predefined for every individual, a simple direct encoding scheme was employed. Every parameter was encoded on the genotype by a

real number in the range 0 to 99, and this was mapped onto the relevant range during decoding. The parameters that were encoded and the ranges onto which they were mapped are as follows:

- 36 connection weights for the leg-controller sub-networks mapped onto the range ± 16.
- 12 cross-body and along-body coupling connection weights mapped onto the range ± 16.
- 36 infra-red sensor neuron to synapse connection weights mapped onto the range -6.5 to 25.5.
- 36 bumper sensor neuron to synapse connection weights mapped onto the range -6.5 to 25.5.
- 36 bias neuron to synapse connection weights mapped onto the range -6.5 to 25.5.
- 9 unit threshold constants mapped onto the range ± 4: 6 for the leg-controller sub-network neurons, 1 for the infra-red sensor neurons, 1 for the bumper sensor neuron and 1 for the bias neuron.
- 9 unit time constants mapped onto the range 0.5 to 5.0: 6 for the leg-controller sub-network neurons, 1 for the infra-red sensor neurons, 1 for the bumper sensor neuron and 1 for the bias neuron.
- 3 input connection weights mapped onto the range ± 16: 1 for the infra-red sensor neurons, 1 for the bumper sensor neuron and one for the bias neuron.

which makes a total of 177 parameters. Thus genotypes were strings of 177 numbers in the range 0 to 99.

Genetic algorithm and genetic operators

The genetic algorithm was an extremely simple generational model with tournament selection and elitism. After evaluating every member of the population, offspring genotypes were repeatedly produced until the next generation was full. To make a new offspring, two pairs of individuals were picked at random from the population and the fittest individuals from each pair (i.e. the winners of the tournaments) were chosen to act as parents. The offspring genotype was then formed from these two parents through a process of crossover and mutation: single point crossover was applied with a probability of 1, and every one of the 177 numbers that made up the offspring had a 0.02 chance of being mutated. A mutation involved changing the number in question by a random amount taken from a roughly normal distribution with a standard deviation of around 18. If the new value was greater than 99 or less that 0 then it was clipped to lie within this range.

5 Experimental results

After removing some initial bugs from the code[2], reliably fit controllers evolved on practically every run within around 3500 generations. This took around 14 hours to run on a Sun Ultra SPARC and simulated over 11 weeks worth of real world evolution. When downloaded onto the real octopod, reliably fit controllers made the robot walk around its environment in a satisfactory manner, turning away from objects that fell within infra red range on both the right and the left hand side and backing away from objects that it hit with its bumpers.

Unfortunately, we must make do with the bald statement of fact that evolved controllers successfully crossed the reality gap. It is not possible to demonstrate it here due to both the nature of the octopod robot itself and the format of the paper. If the robot was equipped with position sensors on each of the legs then data recorded from these sensors as the robot moved around a real-world environment could be used to provide such a demonstration. The robot, however, is not equipped with sensors of

[2] One such bug, spotted by Jerome Kodjabachian, meant that the penalty due to robot instability was effectively applied at random. Surprisingly, even with such a fundamental error in the code, controllers evolved that were able to perform the task perfectly satisfactorily when downloaded onto the robot.

this type and data of the required type is not available. The other form such a demonstration could take, and probably the most natural, is the evidence provided by video footage of the robot wandering around its environment. This cannot, however, be profitably presented as part of a text and pictures document; even if a sequence of stills taken at short and regular time intervals were displayed, this would not be all that informative as to how the legs of the robot moved in the real world unless there were an impractically large number of them.

In lieu of any method of demonstrating how the legs of the real robot moved as it wandered around its environment, the best we can do is to provide a demonstration of how the motor signal patterns to these legs change in response to each of the four sensory scenarios. Figure 7 offers such a demonstration for a typical reliably fit controller that evolved after 3200 generations. From top to bottom, the first eight traces provide a novel representation of the motor signals issued to each leg over the course of an average fitness trial, and the bottom two traces show the resultant velocities of the left and right side of the simulated robot respectively. The best way of explaining how to read the slightly bizarre looking motor traces is to describe how they were generated. At each iteration of the simulation, a short line representing the current motor signal was added to the right hand side of each motor signal trace. As can be seen from the figure, these lines were of various thicknesses and were always drawn from the horizontal centre line of the trace either up and to the left or down and to the left with various different gradients. The thickness of each line represented the vertical angle of the leg relative to the ground as specified by the motor signal in question: the thicker the line, the lower the leg, and the thinner the line the higher the leg. The gradient of the line represented the horizontal angle of the leg relative to the body as specified by the motor signal in question: the further up and to the left, the further forwards relative to the body, and the further down and to the left, the further backwards relative to the body. In this way, although they are perhaps harder to read than other less informative types of trace devised to convey similar information (see [1] for example), each of the motor signal traces of figure 7 represents *both* the vertical and horizontal components of the relevant signal over the course of a fitness test.

From the left and right velocity traces in figure 7 it is evident that the octopod moved forwards, then turned on the spot to the right, then backed up for a period and then rotated on the spot to the left. This corresponds to the order in which the four sensory scenarios arose during the fitness test that gave rise to this figure: no sensors active, left IR sensor active, bumpers and whiskers active, right IR active. Close inspection of the eight motor signal traces reveals:

- In the absence of any sensory activity the robot proceeded forwards using the classic tripod gate. Note that each leg is perfectly out of sync with the leg directly opposite it on the other side of the body.
- In response to activity from either of the two IR sensors, the motor signals sent to each of the legs on the side of the robot furthest from the sensor suddenly became the exact opposite of the signals sent to each of the corresponding legs on the side of the robot nearest the sensor. This made the side nearest the sensor signal go forwards and the one furthest away move backwards.
- In response to activity from the bumpers and whiskers, the robot proceeded backwards using a backwards tripod gate. Note that just before this phase of the simulation was finished, but well after the short-lived inputs to bumpers and whiskers had ceased, the robot paused with all legs down and back for a moment.

When downloaded onto the real robot, these motor patterns and walking gates were clearly and reliably recognizable.

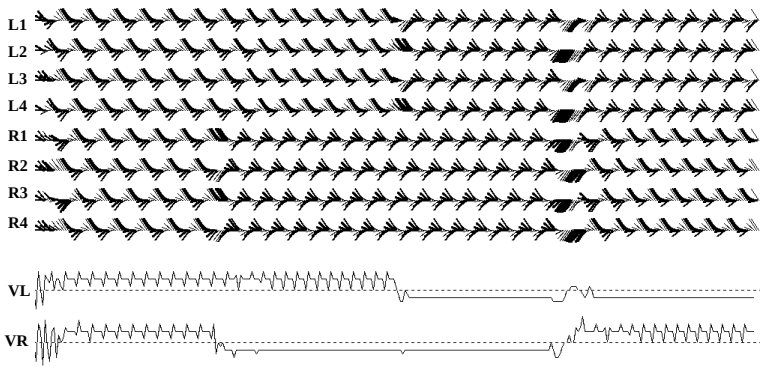

Fig. 7. Each leg controller consisted of six fully connected neurons. The activity of neuron 1 and neuron 2 controlled the horizontal and vertical leg angles respectively.

6 Comments

The minimal simulation used in this paper makes full use of the arguments put forward in [9] to evolve controllers for the octopod robot. Simply put, these arguments state that a minimal simulation need only model the real-world dynamics involved in successful behaviour and no others. This is because the only controllers that must cross the reality gap, if the simulation is to be a success, are precisely those that use these dynamics (i.e. perform the behaviour) and no others. For many robotics setups and behaviours this may not be of any use since the dynamics involved in successful behaviour may be neither obvious ahead of time nor qualitatively different to the rest of the dynamics of the system. For the experiments reported in this article, however, the dynamics of the octopod robot during successful walking and obstacle avoiding behaviour were both relatively easy to identify and *much* easier to model than the dynamics of the octopod robot as a whole. A minimal simulation that modelled these dynamics alone was therefore easy to construct and ran extremely fast when compared to the simulation that would result from attempting to model *all* of the dynamics of the octopod robot within its environment.

Acknowledgements

First, infinitely many thanks to Phil Husbands for editing this paper out of the thesis chapters I left him while I disappeared on the piste. What a fine man. Blame the title on him. Second, many thanks to Phil Husbands, Inman Harvey, Mike Wheeler, Giles Mayley, Joe Faith, Adam Bockrath, Seth Bullock, Jon Bird, Pete de Bourcier, Emmet Spier, Adrian Thompson, Suzy Levy and other members of COGS and the CCNR, past and present, for crucial discussions, debate and help. Third, a big salute to the Brighton beach 4:00am naked winter swimming club, I couldn't have done this without you. This work was supported by a COGS postgraduate bursary.

References

1. R.D. Beer and J.C. Gallagher. Evolving dynamic neural networks for adaptive behavior. *Adaptive Behavior*, 1:91–122, 1992.
2. R. Brooks. Artificial life and real robots. In F. J. Varela and P. Bourgine, editors, *Toward a Practice of Autonomous Systems: Proceedings of the first European Conference on Artificial Life*, pages 3–10, Cambridge, Massachusetts, 1992. MIT Press / Bradford Books.

3. F. Gruau. Automatic definition of modular neural networks. *Adaptive Behavior*, 3(2):151–184, 1995.
4. F. Gruau. Cellular encoding for interative evolutionary robotics. In P. Husbands and I. Harvey, editors, *Fourth European Conference on Artificial Life*, pages 368–377. MIT Press/Bradford Books, 1997.
5. I. Harvey and P. Husbands. Evolutionary robotics. In *Proceedings of IEE Colloquium on 'Genetic Algorithms for Control Systems Engineering', London 8 May 1992*, 1992.
6. P. Husbands and I. Harvey. Evolution versus design: Controlling autonomous robots. In *Integrating Perception, Planning and Action, Proceedings of 3rd Annual Conference on Artificial Intelligence, Simulation and Planning*, pages 139–146. IEEE Press, 1992.
7. N. Jakobi. Half-baked, ad-hoc and noisy: Minimal simulations for evolutionary robotics. In P. Husbands and I. Harvey, editors, *Fourth European Conference on Artificial Life*, pages 348–357. MIT Press/Bradford Books, 1997.
8. N. Jakobi. Evolutionary robotics and the radical envelope of noise hypothesis. *Adaptive Behavior*, 6(2):325–368, 1998.
9. N. Jakobi. *Minimal Simulations for Evolutionary Robotics*. PhD thesis, University of Sussex, 1998.
10. 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*, pages 704–720. Springer-Verlag, Lecture Notes in Artificial Intelligence 929, 1995.
11. J. Kodjabachian and J.-A. Meyer. Evolution and development of neural networks controlling locomotion, gradient following and obstacle avoidance in artificial insects. *IEEE Transactions on Neural Networks*, page (in press), 1998.
12. M.J. Mataric and D. Cliff. Challenges in evolving controllers for physical robots. *Robot and Autonomous Systems*, 19(1):67–83, 1996.
13. O. Miglino, H.H. Lund, and S. Nolfi. Evolving mobile robots in simulated and real environments. *Artifical Life*, 2(4), 1995.
14. S. Nolfi, D. Floreano, O. Miglino, and F. Mondada. How to evolve autonomous robots: Different approaches in evolutionary robotics. In R. Brooks and P. Maes, editors, *Artificial Life IV*, pages 190–197. MIT Press/Bradford Books, 1994.
15. A. Thompson. Evolving electronic robot controllers that exploit hardware resources. In F. Moran, A. Moreno, J.J. Merelo, and P. Chacon, editors, *Advances in Artificial Life: Proc. 3rd European Conference on Artificial Life*, pages 640–656. Springer-Verlag, Lecture Notes in Artificial Intelligence 929, 1995.

This article was processed using the LaTeX macro package with LLNCS style