# 1 Introduction

The artificial evolution of control architectures typically involves the constant and repetitive testing of hundreds upon thousands of individuals as to their ability to behave in a certain way or perform a certain task. In the case of real robots this testing procedure is far from a trivial matter and (with the exception of certain hybrid approaches (Thompson, 1995; Nolfi, Floreano, Miglino, & Mondada, 1994a)) can be done in only one of two ways: control architectures must either be evaluated on real robots in the real world, or they must be evaluated in simulations of real robots in the real world. Both of these approaches have their problems.

As (Ma)9tricetures contar9chitectur5e

This paper puts forwards such a theoretical and methodological basis, albeit at a preliminary stage, and outlines some notable experimental successes using the techniques proposed. Section 2 undertakes a conceptual analysis of how it is possible for control architectures that have evolved in simulation to transfer into reality in the first place. Two conditions are put forwards that must be true of evolved controllers if this transference is to be successful. Section 3 outlines a methodology for building simulations within which reliably fit control architectures are guaranteed to meet these two conditions, and are therefore guaranteed to cross the reality gap. The idea of fast, easy to build minimal simulations is also introduced in this section. Section 4 outlines evolutionary experiments that involve a minimal simulation of a Khepera robot, and section 5 outlines evolutionary experiments that involve a minimal simulation of the Sussex university gantry robot. Finally section 6 offers some conclusions and thoughts for the future.

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

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

### 2.1 What counts as success?

If we are to come up with a general methodology for building simulations for evolutionary robotics it is important to define exactly what we mean when we say that a behaviour has successfully transferred from simulation to reality. In Miglino et al. (1995), the authors look at the fitnesses of control architectures in simulation and compare them to the fitnesses of the control architectures in reality, but as we shall see in section 3.4 this is not always possible and we shall not be using this criterion here. In Jakobi et al. (1995), the authors use a more subjective approach to judge whether control architectures behave qualitatively similar in reality to how they behave in simulation, but again, as we shall see, this is not always possible either. For the purposes of this paper, a control architecture is said to have successfully crossed the reality gap if it successfully displays the behaviour it was evolved to display when down-loaded

---

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

modelling process for the time being), there will be no differences between simulation and reality from the point of view of evolving robot controllers; we can therefore give evolution a free rein, safe in our knowledge that whatever aspects of the simulation evolving controllers come to depend upon, they will also be present in the real world. In practice, however, no matter how comprehensive the model, there will always be real world features that have been left out. Even with extensive and time-consuming empirical validation, simulations built according to this approach can only hope to capture a subset of the totality of possible robot-environment interactions. They should therefore be thought of in the same way as the simulations discussed above.

# 3 A new methodology for building simulations

In the previous section, two properties were put forward that together are sufficient for the successful transfer of robot controllers across the reality gap: they must be base set exclusive, and they must be base set robust. In this section ways of forcing the evolution of these two properties not rely on *how* they arise within this interval (the implementation aspect), but only on the fact that they do (the base set aspect).

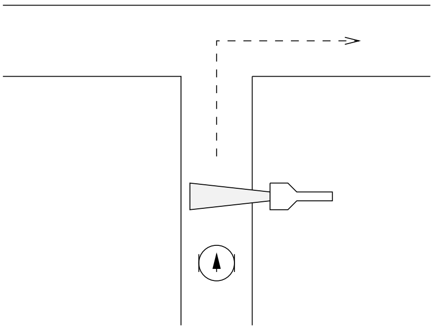


Figure 3: The task in the real world.

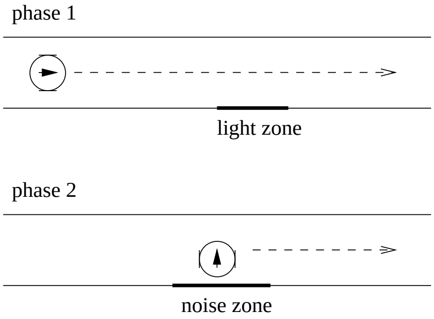


Figure 4: The task in simulation.

in the Evolutionary Robotics literature so far. The behaviour that was decided on is shown diagrammatically in figure 3. As a Khepera robot begins to negotiate a T-maze, it passes through a beam of light shining from one of the two sides, chosen at random. To score maximum fitness points the control architecture must 'remember' on which side of the corridor the light went on and, on reaching the junction, turn down the corresponding arm of the T-maze. This behaviour involves several elements: not only must controllers guide the robot down the corridors without touching the sides and negotiate the junction at the end of the first corridor (simple reactive behaviours both), but they must also involve some internal state that allows them to 'remember' which side the lamp was on so that they can take the correct turning at the junction.

## 4.2 The minimal simulation

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

1. the way in which the robot moves3(w)9h99.3(mh99.3(mh994350000.01(y)-viours.)Tlo)999.959(w)9.75 ion.

At first glance number 2 of the above list seems totally counterintuitive since a T-maze has nothing to do with infinite corridors. However, with respect to the infra-red sensors of a Khepera which have a maximum range of only about 8cm, a T-maze is identical to an infinite corridor almost everywhere. Where a T-maze differs from a corridor, at the T-junction, the interactions between the sensors and the corridor walls were treated as implementation aspects of the simulation, and randomly varied from trial to trial according to the methodology laid out above. In this way, reliably fit controllers were forced to use strategies that depended on the interactions between the infra-red sensors and the sections of the walls of the T-maze that could be regarded as straight and continuous corridor walls, and those interactions alone. First we will describe the way in which the simulation of a T-maze was constructed from two different phases of a simple continuous corridor model, and then we will describe how the corridor model itself was put together.

### Simulating a T-maze with two corridors

Figure 4 shows the two phases of the T-maze simulation. In the first phase, the virtual robot had to travel down a simple corridor where it received a light signal from either one side or the other. After it had travelled a predetermined distance, it was suddenly popped out of the first corridor, rotated through ninety degrees, and popped into the middle of a second corridor for phase two. It then had to chose whether to turn left or right, depending on which

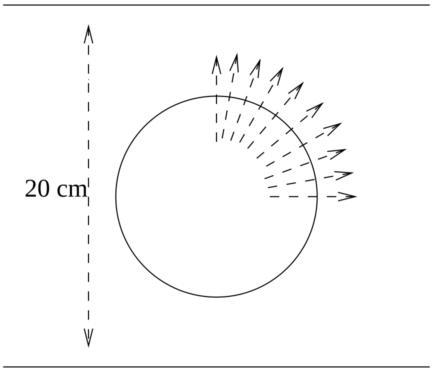


Figure 5: A look up table contains the perpendicular distances to the walls of a 20cm wide corridor for all eight sensors in ten possible orientations.

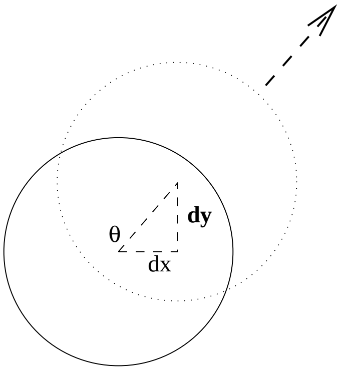


Figure 6: A look up table holds horizontal and vertical increment values for 36 different orientation values and an average speed of 1.

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

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

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

### Simulating an infinite corridor

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

The simulation was updated the equivalent of ten times a second. Figure 6 shows how the new position of the virtual Khepera within its environment was calculated at each iteration. The orientation was used as an index to a look-up-table with 36 pairs of values: horizontal and vertical increments for a Khepera travelling at a speed of 1cm per second. To work out its new position, the values returned from this look up table were multiplied by the average wheel speed in cm per second. The speed of each wheel was calculated directly from multiplying the motor signals by the constant 0.8 cm per motor unit per second. The change in orientation at each iteration was equal to the difference between the distances the two wheels moved divided by the radius of the robot (about 5.2cm). There was no allowance for momentum and the noise inherent in

10cm. For example, if the distance from the robot to a wall was actually 5cm instead of 10cm, then look-up-table values for sensors that pointed at that wall were halved. In this way, the 80 values of the look-up-table were sufficient to find the approximate distance, warped according to the equation given above, from the centre of the robot in any position and any orientation, along the line of any sensor, to the wall of an infinite corridor of any width.

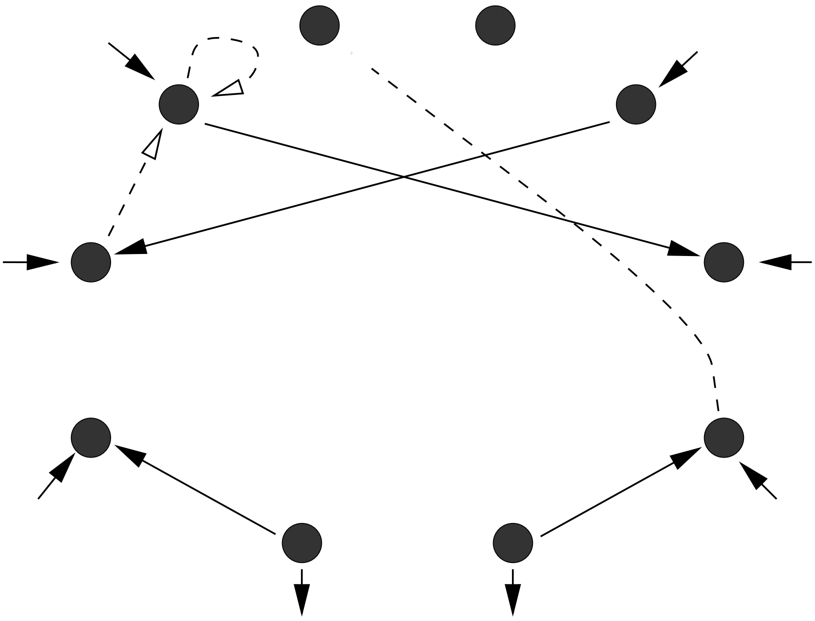
Having ascertained warped distance values *wdv* for each sensor, the actual value that each simulated sensor returned, $V$, was given by a simple linear function:

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

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

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

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



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

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

The genetic algorithm was a steady-state distributed genetic algorithm (Collins & Jefferson, 1991) with a population of 100 individuals arranged on a virtual 10 by 10 grid. At each iteration, a random location was chosen on the grid and a breeding pool constructed from the nine individuals of the 3 by 3 square centred on that location. Two probabilistically fit parents were chosen from this breeding pool according to a linear rank-based selection procedure, and an offspring constructed by a process of crossover and mutation. This offspring then replaced a probabilistically unfit member of the same breeding pool according to an inverse linear rank-based selection procedure. Single point crossover was applied with probability 0.7 and the expected number of mutations per genotype, according to a Poisson distribution, was 2. At each offspring event, not only was the offspring's fitness evaluated, but both parents were reevaluated as well.

## 4.4 Experimental results

Figure 7 shows a typical example of the sort of neural network that consistently evolved within around 1000 generations (where a generation was taken to be 100 offspring events). This is the simulated equivalent of $300 \times 15 \times 10 \times 100 = 45000000$ seconds or over 17 months of continuous real-world evolution, and takes around 4 hours to run as a single user on a SPARC Ultra. The network reliably achieved near-optimal fitness within the simulation. In order to see whether it would successfully transfer across the reality gap, the network was do999.349(cedu(do99wn385Td(lig)TJ-30011rT1c.00(tak)1000e)-1p,)Tj22kgenerati

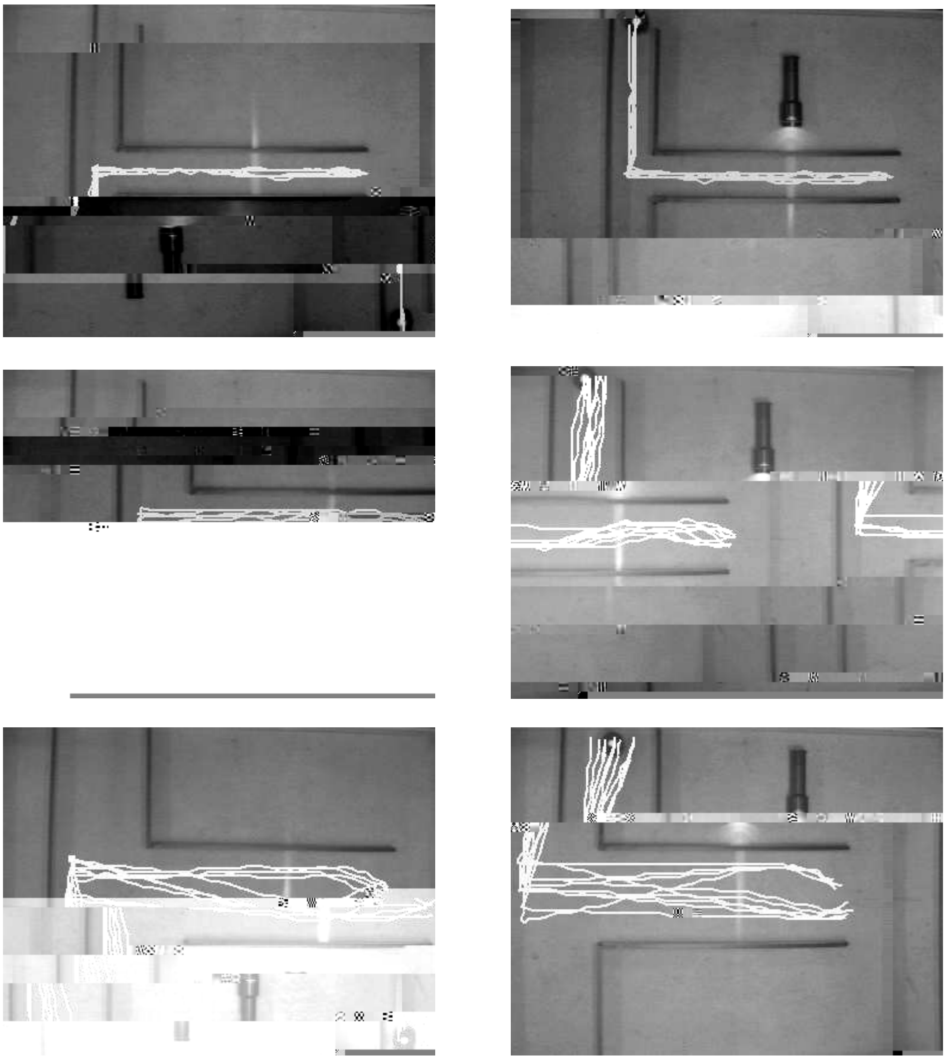


Figure 8: These six pictures together show the paths taken by a Khepera robot in sixty consecutive trials of the control architecture shown in figure 7. These sixty trials were performed in consecutive batches of ten, and each picture shows ten trials for a particular corridor width and torch orientation.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

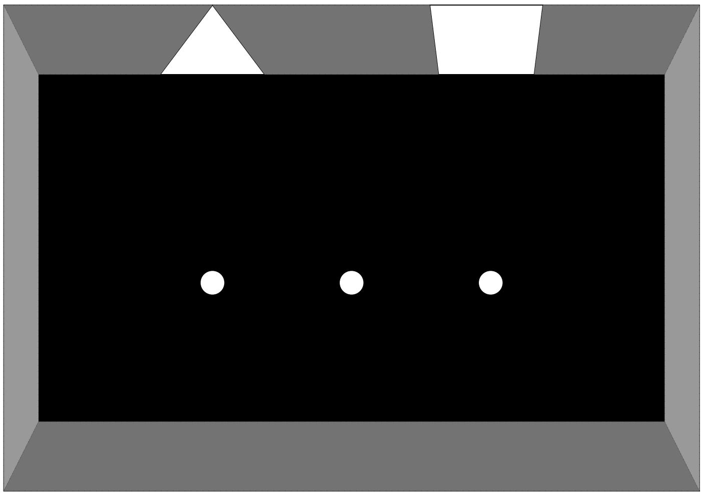


Figure 11: A diagrammatic view of the gantry arena from above showing the four possible starting positions of the gantry robot. The dashed line in front of the triangle marks the area that the gantry must reach in order for a trial to count as a success for testing purposes.

## 5.1 The aim

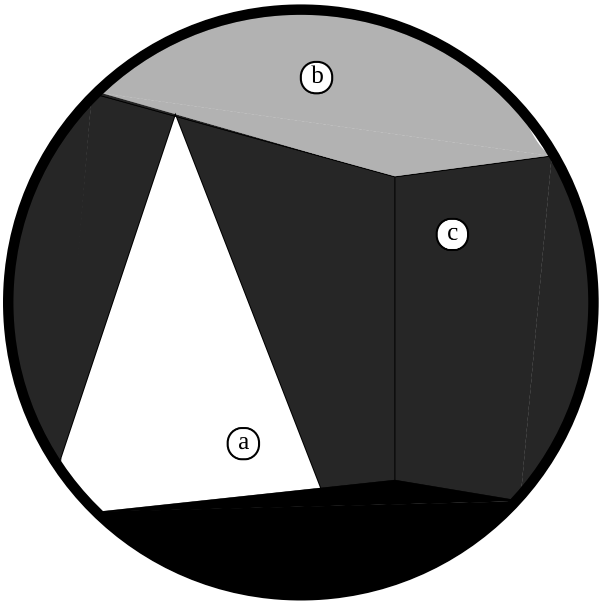


Figure 12: A typical image returned by the camera of the gantry robot. The robot is facing the corner of the arena and the triangle can be seen on the left. The white

to a simulation, since the average value of each circular visual field in Harvey et al. (1994) was just the average value of 25 randomly sampled pixels from within the field. A simulation of either, therefore, must contain a model of how specific pixels of the camera image acquire values in response to the orientation and position of the robot within its environment.

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

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

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

1. The way in which pixels of the camera image, that project onto the walls of the arena, return grey-scale values within certain intervals: 14 to 15 for pixels that project onto either the triangle or the square, and 0 to 13 for pixels that project onto the walls of the n it

motor unit per second. In addition there was also a momentum term, $m$, such that at each iteration, the increment $\delta v$ to each wheel speed $v$ in terms of the required wheel speed $u$ was:

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

This momentum term was added for the simple reason that in the case of the gantry,

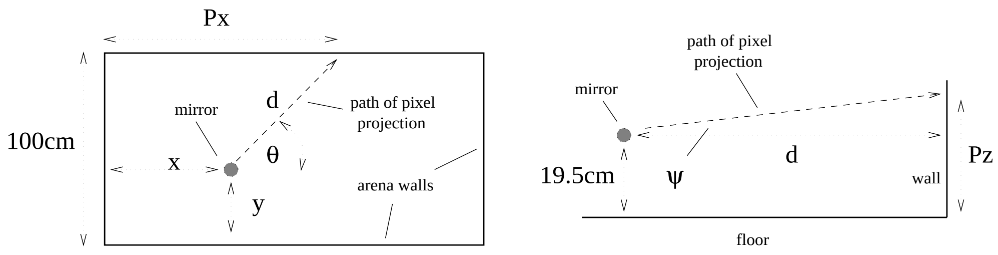

Px
100cm
mirror
d
path of pixel projection
x
θ
y
arena walls
path of pixel projection
mirror
19.5cm
ψ
d
Pz
wall
floor

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

either the ceiling or the floor. If $Pz$ was between 0cm and 22.5cm it was judged to have projected onto a wall. If the wall in question was the one with the triangle and the square on it, then simple geometric relationships between the coordinates of the pixel projection point and the vertices of the two shapes were used to find if the pixel projection point lay inside either of the shapes. At every iteration, a random deviate in the range ±1.2 grey-scale units the coordinates was added to eachTJ-348..349(as)-13999.pathos.3(to)-13999.3(e12-15ue.f55.9

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

3.

- The momentum term, $m$, of equation 5.2 was randomly set at the beginning of each trial to a value between 1 and 4.
- Random offsets of between ±0.5cm per second were generated at the beginning of each trial, and added to required wheel-speeds during position update calculations.

Together these random variations ensured that reliably fit control architectures were able to cope with a wide variety of slightly different robot-environment interaction models. Included in this range were models that involved misshapen and mal-aligned mirrors as well as noisy and unpredictable motors - such as the model instantiated by the real gantry robot.

## 5.3 The evolutionary machinery

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

Figure 14 shows a typical example of the type of control architecture used in the experiments reported here. Functionally they are very similar to those used in the Khepera experiments described in section 4: weights on links are in the range ±2 and thresholds are in the range 0 to 1. The activation function of every unit *including* the motor neurons was that of equation 4.3. In addition to a genetically determined number (with a maximum of 3) of connections to each neuron from other neurons in the network, neurons could also receive normalized input, in the range 0 to 1, from a camera image pixel. Motor signals were calculated from the output values of the four larger corner neurons of figure 14 according to the relation $signal = 2 \times (A_1 - A_2)$, where $A_1$ and $A_2$ are the output values of the appropriate forwards and backwards neurons. The whole network including inputs and outputs, and therefore the whole simulation, was updated at a speed of 10 times per simulated second.

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

Development took place in a two dimensional space, with the position of each neuron (apart from the four motor neurons, see figure 14) genetically determined within that space. Each link within the network to any particular neuron was genetically specified in terms of the desired position of the neuron from which the link originated. The nearest neuron to this desired position, within a certain radius, was then alloted as the originator of the link. If no neurons lay within this radius, which was set at around

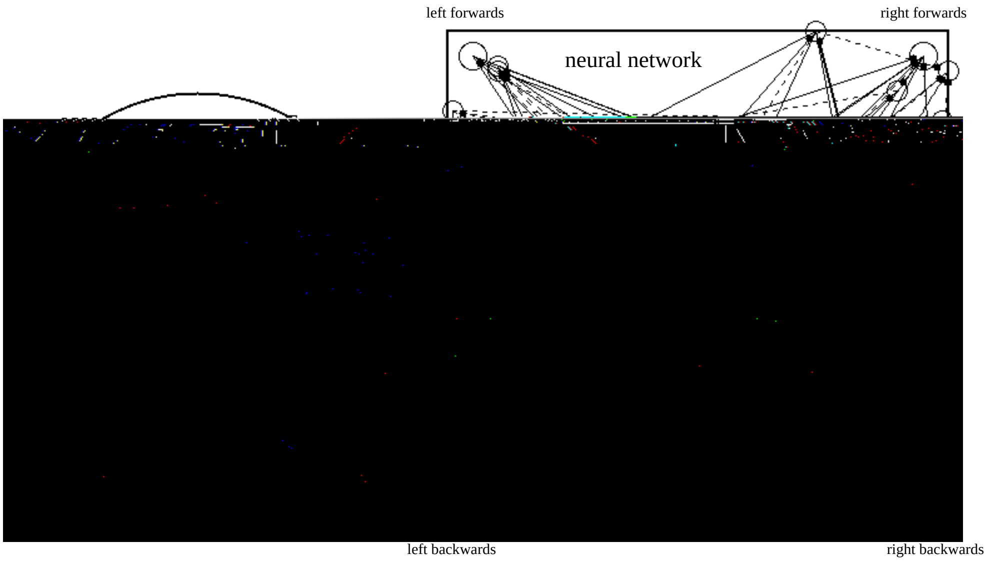

left forwards
right forwards
neural network
left backwards
right backwards

Figure 14: An example of a typical neural network evolved for the triangle/square discrimination task. Ontherypical

every member of a population of 100 individuals, the fittest 25 were used to produce the next generation by randomly picking parents and producing offspring until the new population was full. Crossover was applied with a frequency of 0.7 and the expected number of mutations per genotype, according to a Poisson distribution, was 1. There was a probability of 0.02 at each offspring event that a random gene would be introduced into the offspring genotype, as well as a probability of 0.02 that an already existing gene would be deleted.

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

### 5.4 Experimental results

Figure 14 shows a typical example of the sort of network that evolves to be reliably fit circumstances. This may have been due to freak noise, but may also have been due to a mechanical or software error.

With

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

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

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

Collins, R., & Jefferson, D. (1991). Selection in massively parallel genetic algorithms. In Belew, R. K., & Booker, L. B. (Eds.), *Proceedings of the Fourth Intl. Conf. on Genetic Algorithms, ICGA-91*, pp. 249–256. Morgan Kaufmann.

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

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

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

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

Husbands, P. (1997). Personal communication..

Husbands, P., & Harvey, I. (1992). Evolution versus design: Controlling autonomous robots. In *Integrating Perception, Planning and Action: Proceedings of the Third Annual Conference on Artificial Intelligence, Simulation and Planning*, pp. 139–146. I.E.E.E. Press.

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

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

Jakobi, N. (1996). Encoding scheme issues for open-ended artificial evolution. In Voigt, H.-M., Ebeling, W., Rechenberg, I., & Schwefel, H.-P. (Eds.), *Proceedings of the Fourth International Conferrence on Parallel Problem Solving iencSoei7226xue*

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

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

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

Michel, O. (1995).