Evolutionary and Adaptive Systems MSc.

Artificial Life Project Report

Evolutionary Robotics Simulation using a CTRNN

By Mike Blow Candidate 80761

27.12.2003

Introduction

Robot control has become a topic of great activity and some controversy of late with the original A.I. 'sense-plan-act' approach being challenged by Brooks' subsumption architecture **[3]** and evolutionary robotics. The latter potentially offers much by letting evolution solve the problems of control for us, but requires many generations of robots to be evaluated and can take a prohibitively long time. The obvious compromise is the evolutionary robotics simulator, where the evolutionary runs can be simulated in a fraction of the 'real world' time on a computer. This project aims to create a robot simulator and use it to evolve a robot capable of approaching and then staying near a light.

Evolutionary Robotics

Evolutionary Robotic aims to solve a problem via incremental steps towards an optimal (or near optimal) solution. There have been several successes in the field of robot control using this approach **[4,9]** and also some interesting work in evolvable morphology **[8]**. In all of these experiments an initial population of genotypes is created, usually randomly although possibly constrained in some way (for instance a symmetrical neural network may be specified if it is known that this is necessary. Such constraints if properly applied can speed the evolutionary process by reducing the size of the evolutionary search space **[6]**. Each genotype is used to create a phenotype and these phenotypes are then evaluated with respect to the required task. The most successful individuals are then reproduced using crossover (where genes

from one parent replace genes from the other in the child genotype) and mutation (randomly altered values in the child genotype). Gradually the fitness of the population increases until it reaches an optimal value. A genetic algorithm of this sort is not always guaranteed to find the best solution but will usually find a good one given the right starting conditions and fitness evaluation criteria. One thing that has been learned is that choosing the right fitness function is extremely important, and it is not always trivial. It can take several attempts to get right **[4]**, and the implications of any particular rule need careful consideration.

The genetic approach to robotics can be seen to have a lot of potential, especially considering the complexity of a robot's task. It is impossible to anticipate the vast number of situations caused by existing in, and interacting with, the real world. The solution to this problem historically has been to constrain the environment. In theory coping with the real world is just a scaling up of this approach – after all the real world still has its constraints - but it has become apparent that the problem becomes all but intractable and to achieve robots that can reliably cope in this environment a new approach is needed.

The Robot Model

Each robot was simulated as a differential steering platform with two light sensors and two motors after Jakobi's minimal Khepera robot simulation **[6,7]**. The radius of the robot and the motor constant (the amount of movement for maximum motor input) were both taken to be 1. The sensors were each modelled as having an 80 degree field of vision placed at 20 degrees off each side of the front of the robot. They therefore had an overlap field of 40 degrees where both sensors would be stimulated simultaneously, and 'peripheral' vision of 60 degrees each side where only one sensor would be stimulated (*fig. 1*).

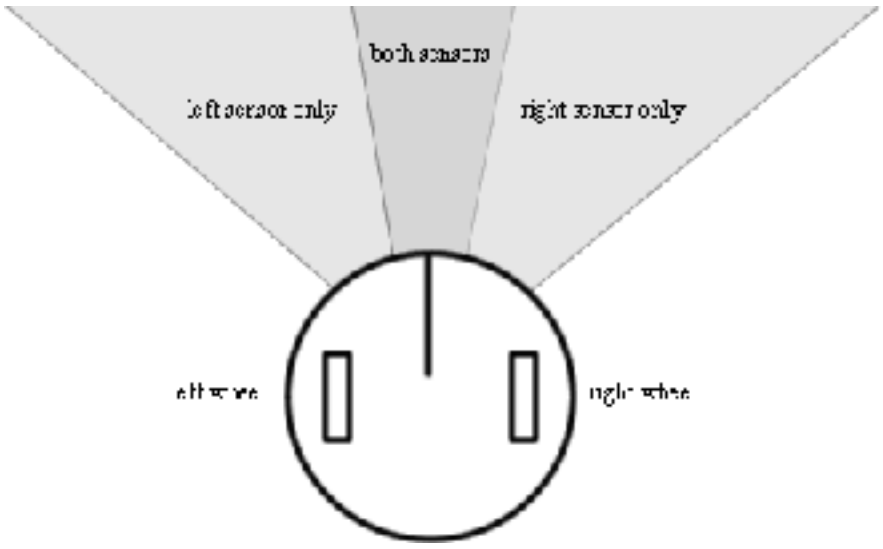


fig. 1 the robot model

The sensors were not modelled to be on the surface of the robot and effectively originated from its centre. They were modelled as having consistent activation over their whole field of view, and receive light according to the inverse square law of distance:

perceived brightness = source intensity / radius from light^2

A luminance value for the light was chosen so as to give a small amount of stimulation to the sensors at the average starting distance from the light. This effect could also be achieved by incorporating a variable gain value into nodes 0 and 1 attached to the sensors, although this was not attempted. The motor values were the outputs of nodes 3 and 4. During a run the outputs of the nodes were not constrained so the motor values were mapped using a sigmoid function to between –5 (full reverse) and 5 (full forward). Random noise was added to the sensors and motor signals at a level not exceeding 10% of the maximum value of the device.

The Network

A Continuous Time Recurrent Neural Network (CTRNN) was used as a controller for the robot. CTRNN nodes are governed by the equation

$$\frac{dx_i}{dt} = \frac{1}{\tau_i}\left(-x_i + \sum_{j=1}^{N} w_{ji}\sigma(x_j + \theta_j) + I_i\right) \qquad i = 1,2,\ldots N$$

Where:
$\mathbf{x}_i$ is the output of node *i*
$\boldsymbol{\tau}_i$ is 'tau', the learning rate of node *i*
$\mathbf{w}_{ji}$ is the weight of the connection from the *j*th to the *i*th node
$\mathbf{I}_i$ is the external input to node *i*

And $\boldsymbol{\sigma}$ represents the sigmoid activation function:

$$\frac{1}{1 + e^{-(x_j - \theta_j)}}$$

where:
$\mathbf{x}_j$ is the output of node *j*
$\boldsymbol{\theta}_j$ is the bias of node *j*

The inclusion of the time dependent term '$\boldsymbol{\tau}$' gives this kind of node a form of memory and with it the potential for behaviour beyond the merely reactionary. CTRNN networks can perform complex oscillatory and even chaotic behaviour, and have plausible analogies to natural neurons where the output can be thought of as the membrane potential and the sigmoid function is associated with its firing frequency. A detailed study of CTRNNs can be found in **[1]**.

The network consisted of 5 nodes, each with a connection to all the other nodes including itself. Node 0 was connected to the left light sensor, node 1 to the right light sensor, node 3 to the left motor and node 4 to the right motor. (*fig. 2*). The nodes had fixed weights and biases determined by the robot's genome. The output of each node was integrated using the Euler method over timesteps of 0.1 simulated seconds (N.B. in this report all quantities are assumed to be simulated unless stated otherwise).

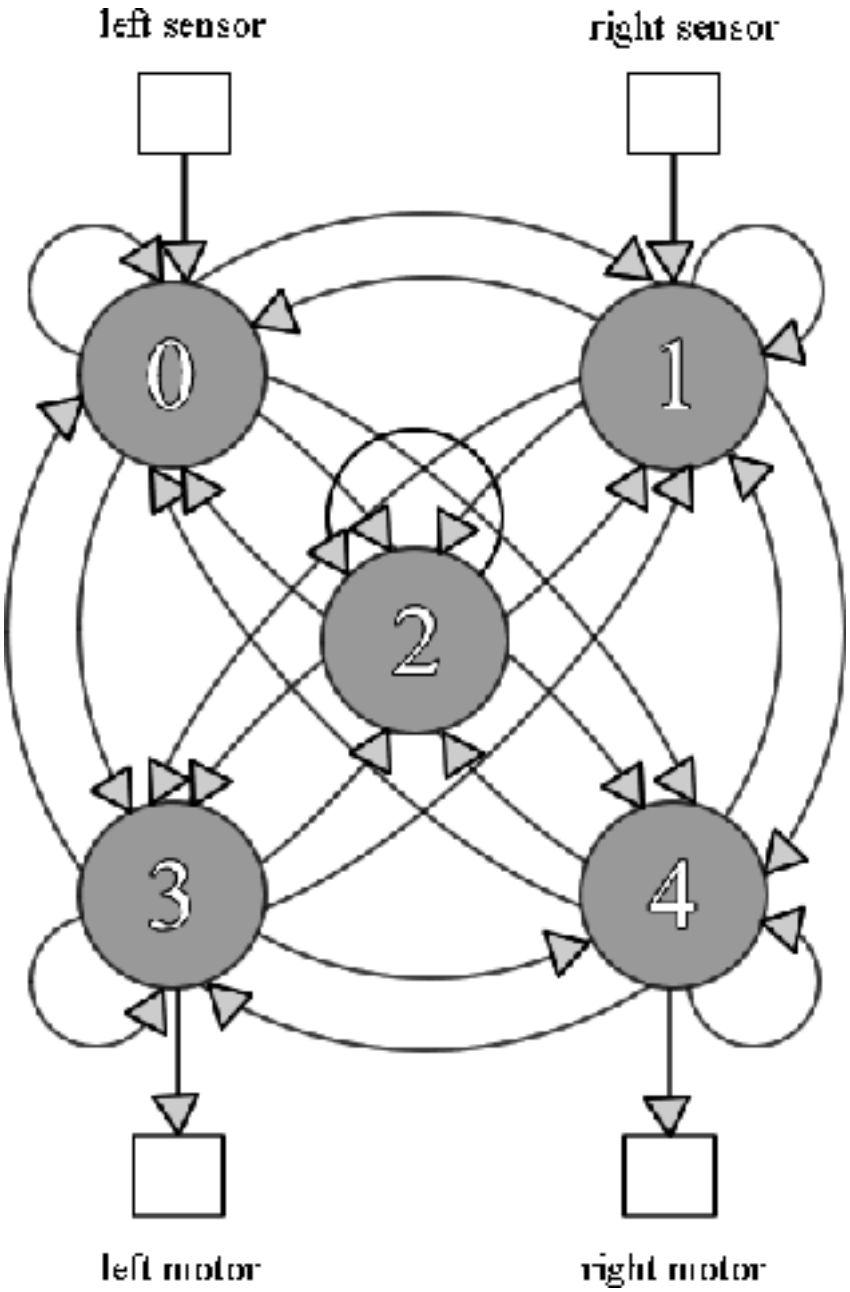


fig. 2 the network and its external connections

The Simulator

The simulator was based on Jakobi's minimal simulator **[6]**. The simulation models a simple differential-steer robot on an infinite plane. No real world physics were used apart from the inverse square law applied to the intensity of the light received at the robot's sensors.

Method

Two arrays of 36 distance values are used, index by the current orientation of the robot. These values correspond to the increment the robot will move in the x and y planes for a given orientation at a speed of 1 centimetre per second. The values returned from the array are multiplied by the average wheel speed, which is calculated as the motor activation value multiplied by the motor constant (the distance the robot will travel for 1 unit of motor input, in this case taken to be 1cm). This gives an increment in the x and y planes that is used to plot the new position of the robot. The change in orientation is given by the distance between the two wheels moved divided by the radius of the robot. In this case for simplicity the radius was taken to be 1, although some experimentation was done to ascertain the change in behaviour given a larger or smaller radius.

Implementation

A graphical implementation of the simulator was created (see colour slide, Appendix A), consisting of three windows:

Arena

The arena displays the progress of the robot. The robot is represented by a grey circle with a line to represent its orientation, and in *fig. 3* can be seen heading towards the light in the top left corner. The light is represented in the arena by a yellow circle surrounded by a yellow ring. The ring represents the scoring zone where the robot is close enough to the light to gain fitness. The paths the robot has taken in each run are represented with white dots. Each robot has a number of trials, and there may be several paths displayed as they all remain on the screen until a new robot is evaluated (this is why there are several trails on the right of the screen in *fig. 3*). This makes it easier to visually compare the robots behaviour with lights in different positions and distances.

NodeView

The node view window displays a graphical representation of the neural net, and sensor and motor values at each step (*fig. 4*). Lines are drawn to indicate the node connections but these are simplified for the sake of clarity. The value of each element is represented by its colour, white for 0, increasingly red for positive values and increasingly blue for negative (except for the light sensors which only create positive output). Using this view it is easy to see the state of the network at any time and also the changes due to external input from the sensors. It is also possible to see the learning rate of the nodes in the time it takes for a node to return to its original colour after perturbation by an external input.

StatsView

This window displays the values of the sensors, nodes and motors at every timestep in numerical form (*fig. 4*).

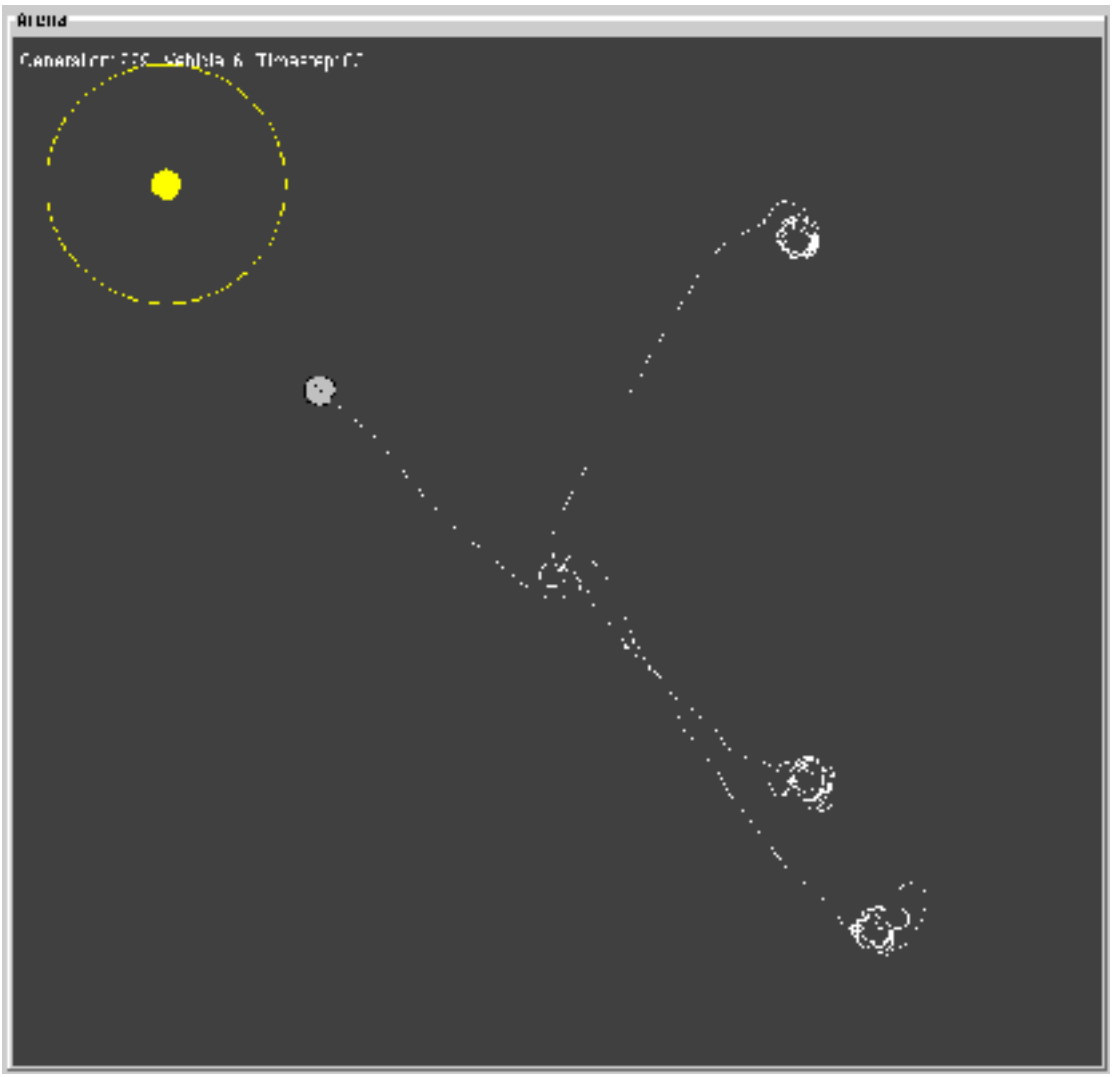


fig. 3 the simulator's Arena window

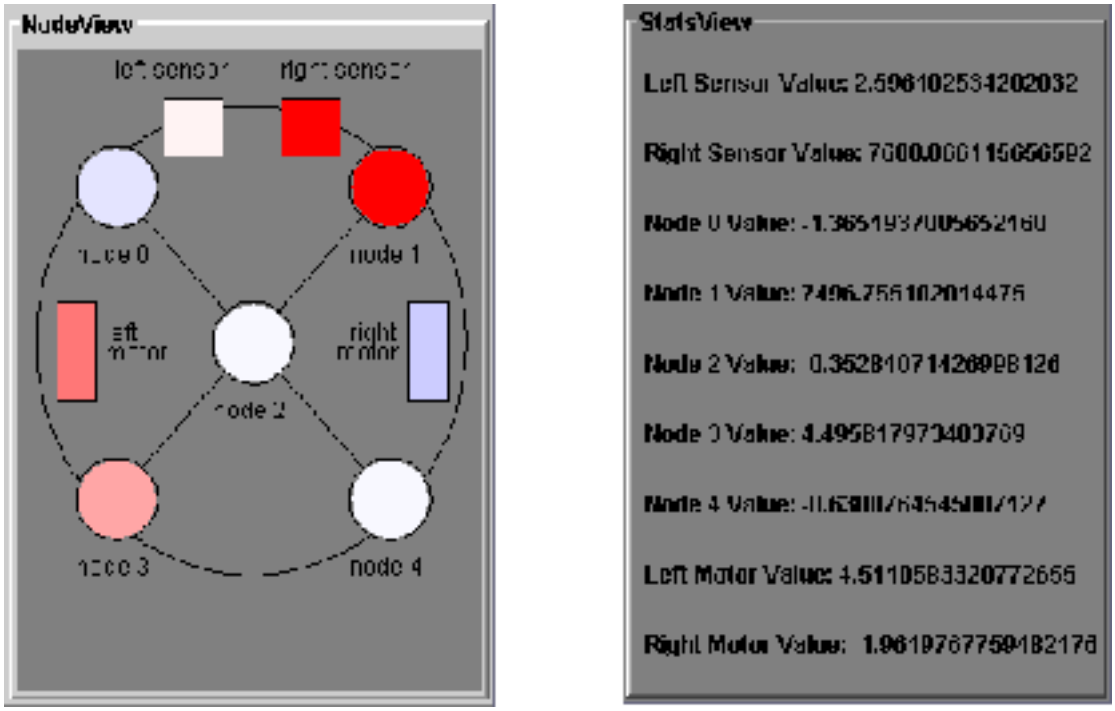


fig. 4 the simulator's NodeView and StatsView windows

All windows are double-buffered. This increases graphical clarity and eliminates flicker, however it does slow down the drawing process significantly so could be turned off for use on slower machines. The graphical display also significantly slows the evolutionary process, so provision was written into the code to turn off the graphical display using a command-line argument. At the end of the run the simulator saves the genomes of the last generation to a text file, and these can then be loaded into the simulator and assessed visually.

The Genetic Algorithm

A generational GA was used with fitness-proportional selection. Initially in both experiments a population of 50 robots was evolved for 500 generations with elitism. Better results were obtained by evolving a population of 200 robots for 1000 generations with no elitism, presumably because the wider spread of initial genomes and lower selection pressure enabled the GA to cover more of the search space. These are the results that are presented later in the report.

Genome

The genome described the controller for each robot by determining the values of the learning rate, bias and connection weights for each node (*fig. 5*). A genome consisted of one NodeDescriptor object per node, each holding seven pieces of information in the form of (initially random) real values:

tau: learning rate of this node, 10 raised to the power of value in the range –1 to +2.
bias: bias of this node, in the range –2 to +2.

weight 0: weight of the connection from node 0, in the range –2 to +2.
weight 1: weight of the connection from node 1, in the range –2 to +2.
weight 2: weight of the connection from node 2, in the range –2 to +2.
weight 3: weight of the connection from node 3, in the range –2 to +2.
weight 4: weight of the connection from node 4, in the range –2 to +2.

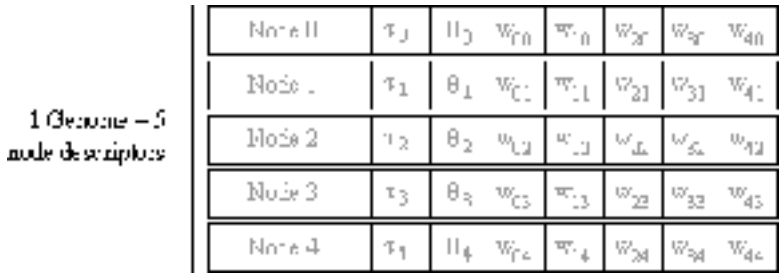


fig. 5 a graphical view of the genome

This genome format assumes a fully connected network but could be modified to represent a network with fewer connections. The value for the learning rate was used as an exponent in order to give a wide range of values, in this case 0.1 to 100 simulated seconds. The rate was capped at a minimum value of 0.1 so as not to be less than the timestep value used to integrate the network. This situation can cause instability in the CTRNN node equation and an exponentially growing oscillating output and should be avoided.

Interestingly a provision to create symmetrical networks was written into the code where node 1 and 4's values were copies of node 0 and 3's respectively. In practise the fully evolved robots performed very similarly to non-symmetrical ones, although crossover and mutation were not constrained to be symmetrical and so presumably the symmetry was lost in the evolutionary process. Nevertheless experiment 1 produced some interesting results in this regard.

Fitness Function

As has been stated before the fitness function is of utmost importance as it is what drives the successive generations towards the required goal. In this case there are two objectives; to approach a light source and once there to stay near it for as long as possible. With this in mind a two-part fitness function was used, modelled on **[5]**:

part 1: points were awarded to the robot for ending the run nearer the light than it started, using the formula:

fitness 1 = 1- (final distance to light / initial distance to light)

This equation returns a value between 1 and 0. The nearer the robot was to the light the higher the score. If the robot finished the run further from the light than it started it was given 0.

part 2: points were awarded for the amount of time of the run the robot managed to spend within a distance of 8 body radii from the light.

fitness 2 = time spent near light / total runtime

This equation also returns a value between 1 and 0 - the more time spent nearer the light the higher the value. The two results were added to give a combined fitness between 0 and 2 with a 50% weighting for each result. Each robot's final fitness value was an average of 5 trial scores.

It is interesting to note that the obvious best strategy – head in a straight line towards the light – is not explicitly defined. However upon further reflection it is clear that it is *implied* by the second rule, in that the quicker the robot gets to the scoring area the higher its fitness will be. This is an important point as it demonstrates that fitness rules can have 'secondary' effects that require careful consideration.

Selection

Fitness-proportional selection was used with the parents of each child being chosen from the whole population. At the end of a generation the individuals were linearly ranked in order of increasing fitness using the algorithm:

$$\mathbf{Fitness}(\mathit{Pos}) = 2 - \mathit{SP} + 2\cdot(\mathit{SP} - 1)\cdot(\mathit{Pos} - 1) / (\mathit{Nind} - 1)$$

where:
SP is selection pressure in the range 1 to 2. A selection pressure of 2 was used in these experiments.
Pos is that robot's position in the rank.
Nind is the size of the population.

The robots were then given a probability of being chosen as parents proportional to their fitness ranking using the equation:

$$\mathbf{Prob}(\mathit{Pos}) = (1/\mathit{Nind})*\mathbf{rank}$$

This method ensures that the fitter individuals have more chance of being chosen as parents and thus increases the average fitness of the population in the next generation. Elitism was not used so as to reduce the chance of the population converging too rapidly on a non-optimal solution.

Breeding

The child genes were initially a clone of one parent's genes. A random crossover operator with a probability of 50% was then used to replace certain genes with those from the other parent. In addition each gene had a 10% chance of being mutated, in which case 0.1 was randomly added to or subtracted from the real value held by the gene. The values were constrained to never exceed 2 in the positive direction and –2 in the negative direction, apart from in the case of the learning rate. This was capped at –1 in the negative direction so the final value was not less than the timestep.

The Tasks

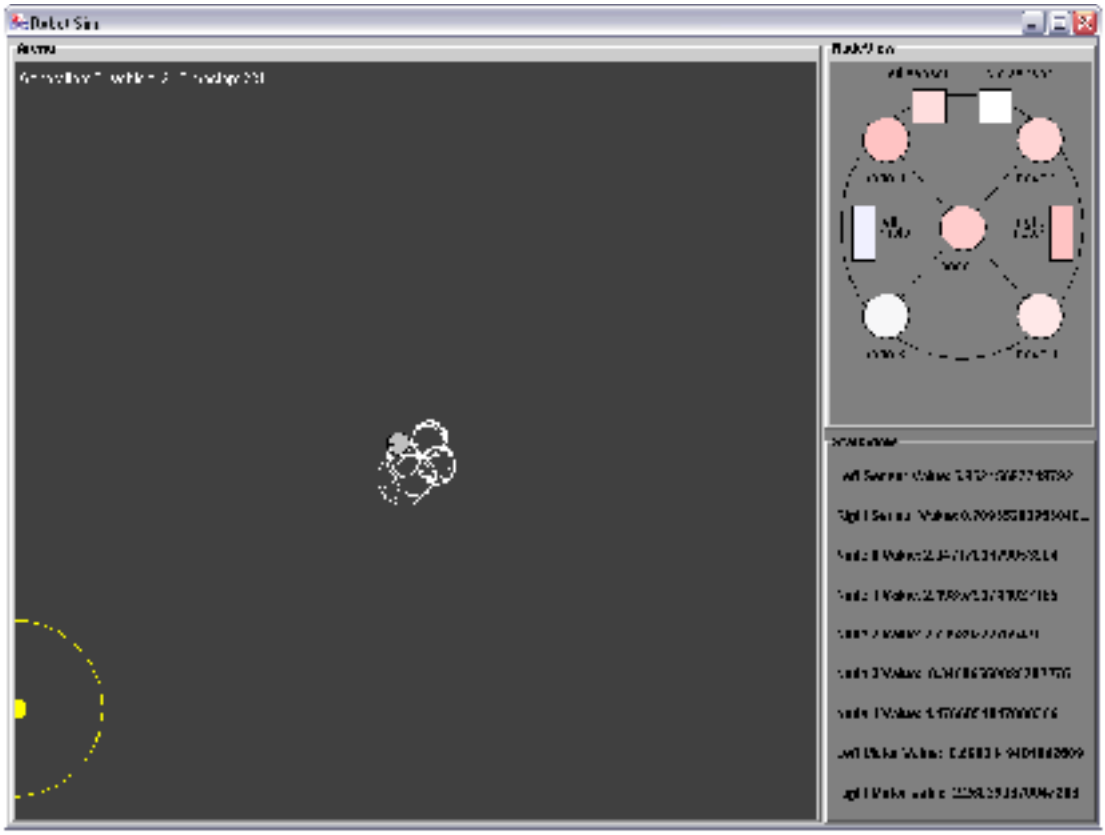
The first task set in this experiment was a relatively simple one – the robot, starting from the middle of the test arena but with a random orientation, should approach a randomly-placed simulated light and then stay within a predefined radius of it. This task is very similar to one described by Braitenberg in **[2]**, and can be achieved by a vehicle without a neural network. However it can be used to demonstrate the principles of evolutionary robotics and is a good basis to build on.

Once this task had been completed a new task was devised to attempt to test the time-variant behaviour of the CTRNN network. The robot would be allowed to spin in place but not move for 4 seconds, after which the sensors would be turned off and the robot allowed to roam freely for another 4 seconds. The question was whether the robot could 'remember' the position of the light and still approach and circle it with no input from the sensors.

Results

Experiment 1

A population of 200 robots was evolved for 1000 generations. Initially the robots tended to spin in circles of various diameters (*fig. 6*).



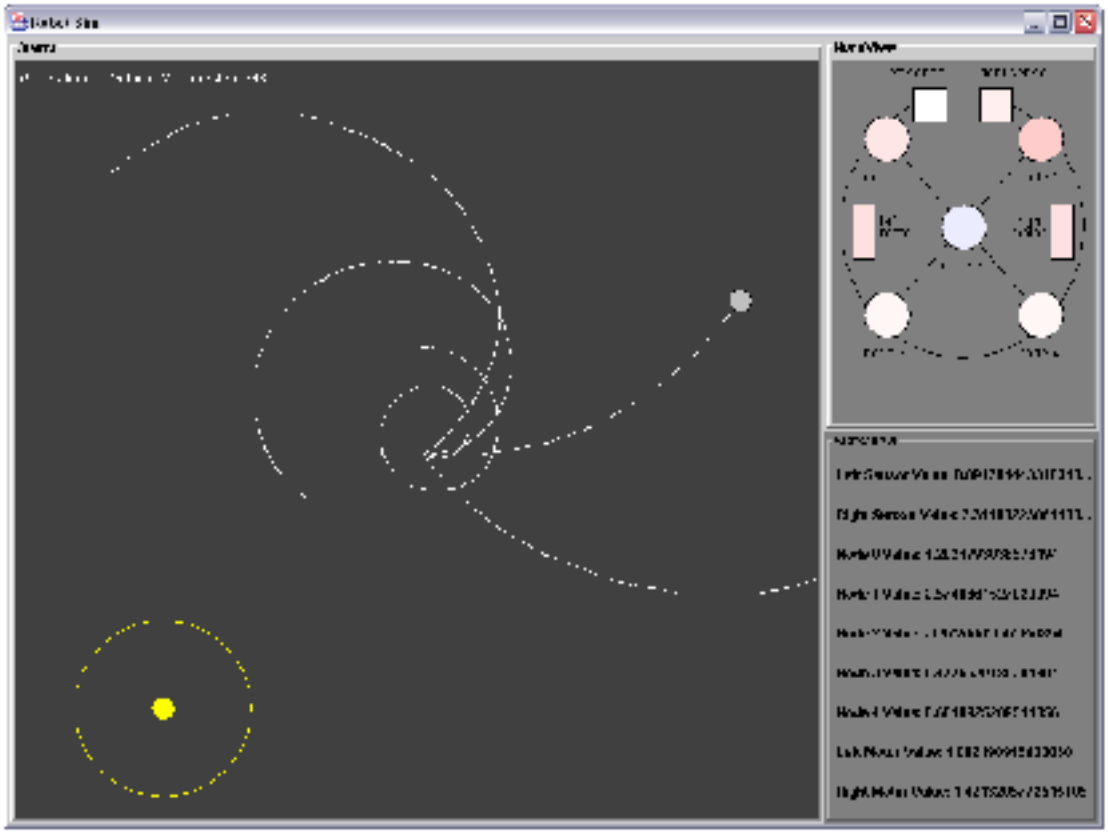


fig. 6 examples of robot behaviour at generation 0

Each robot was awarded a total fitness averaged over 5 assessments of 5 seconds each. At the end of the run a robot had evolved that displayed very effective light seeking behaviour (*fig. 7*). It would spin until it saw the light, then drive fast straight towards the light and once there drive in very small circles passing through the centre of the light as often as possible. This is a good strategy as:

1) the robot reaches the light quickly ensuring it can spend maximum time within the scoring zone
2) the robot has a high chance of being very close to the light when the simulation ends thus maximising its distance score.

It is interesting to note that the second fittest robot in this run had similar tactics but was not as good at keeping to a tight circle around the light (*fig. 8*). The genome values for the fittest controller are also shown in *fig. 7*. Note the similarity between some of the values of nodes 0 and 1, and 3 and 4. This genome has evolved a semi-symmetrical form from totally random initial values. This accounts for the fact that the robot can drive in a straight line but also means that, in theory, it would be resilient to inversion of the visual field as investigated in **[5]**.

	node 0	node 1	node 2	node 3	node 4
tau	0.24	0.11	0.56	0.1	0.1
bias	-2	-1.9	0.77	-0.47	0.79
conn weight from node 0	-1.19	-1.9	-1.69	-1.9	2
conn weight from node 1	-0.89	-0.79	0.56	2	2
conn weight from node 2	-1.49	-1.9	-0.06	1.37	-1.21
conn weight from node 3	-0.99	-1.49	0.93	1.66	-0.98
conn weight from node 4	-1.69	-1.9	-0.99	0.53	-0.28

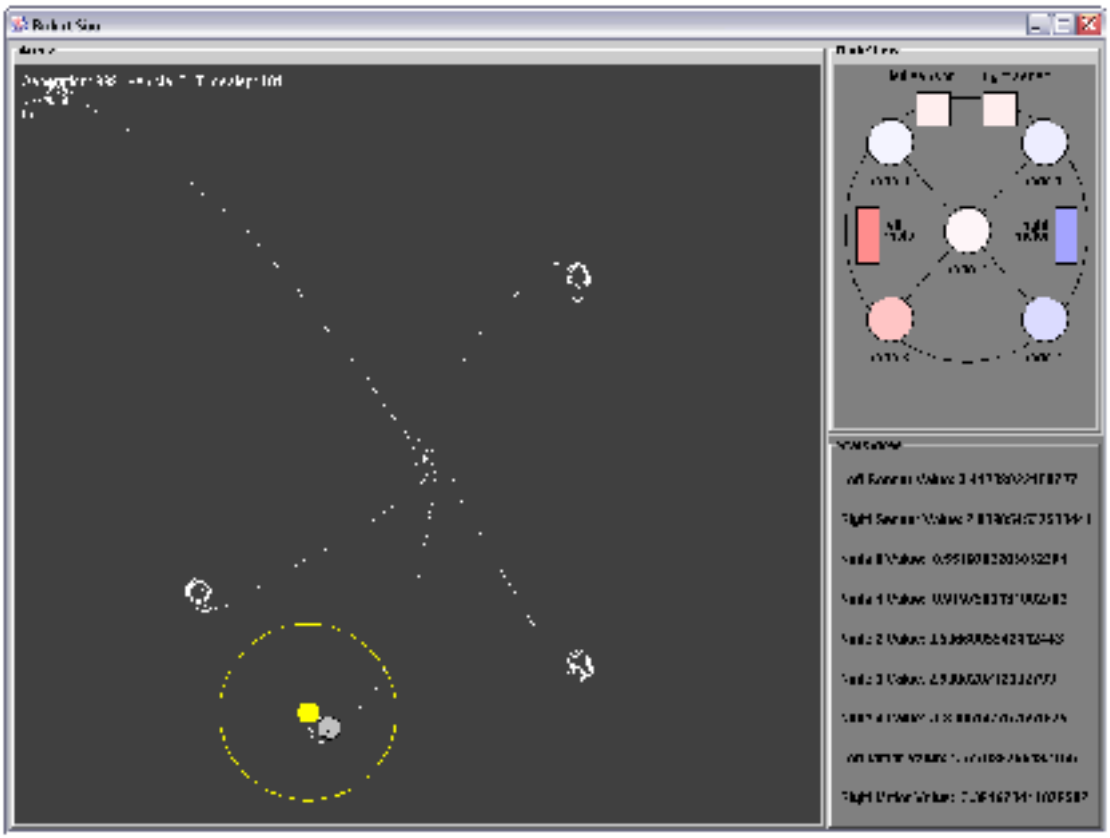


fig. 7 the genome and behaviour of the fittest robot

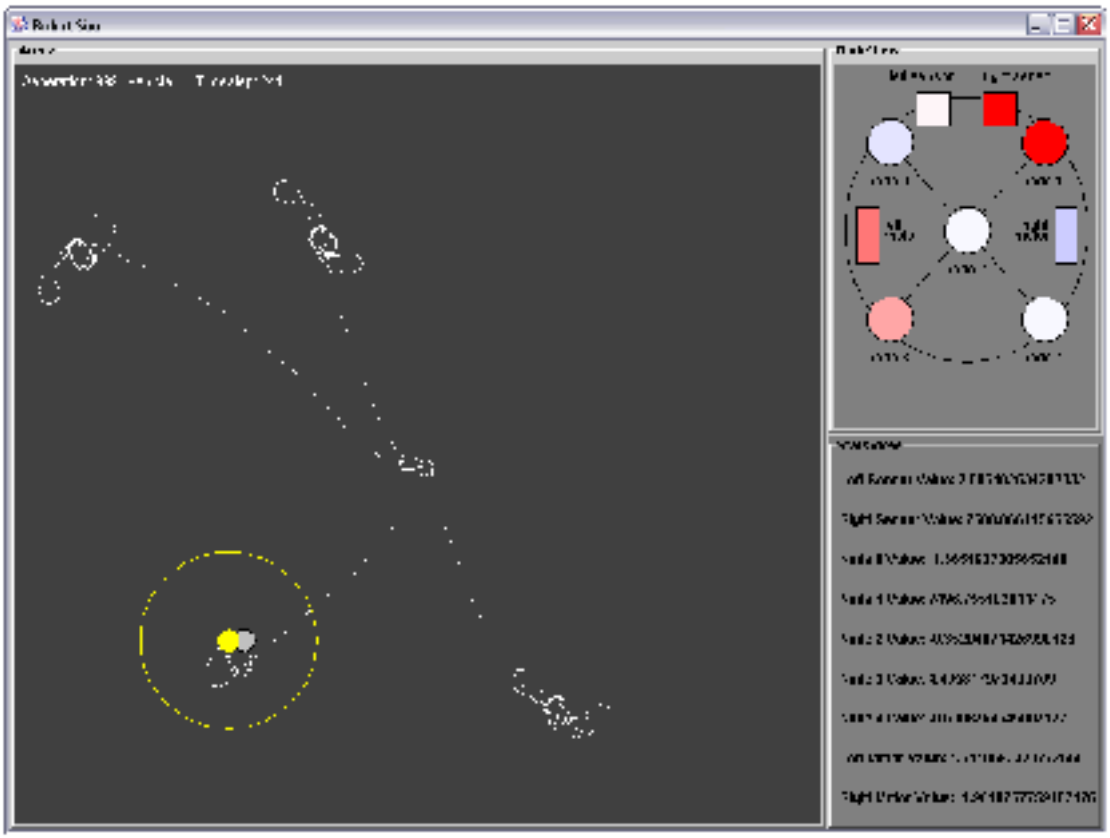


fig. 8 the behaviour of the second-fittest robot

A fitness graph from a typical run is shown in *fig. 9*. The top line is the maximum fitness in each generation and the lower line the average fitness of the whole population. From this it is apparent that the robots reached an maximum fitness level by about generation 500 and did not significantly improve after this. It is also apparent that the maximum fitness sometimes decreases once this level is reached. The two points to take from this graph are firstly that the number of generations could have been halved with little effect on the outcome and secondly that it may be worth trying elitism to retain the fittest individual. Having said this, there is a certain amount of luck involved in the fitness function used (e.g. how close the robot is to the light when the evaluation finishes) which means the same individual will rarely get the same fitness score twice.

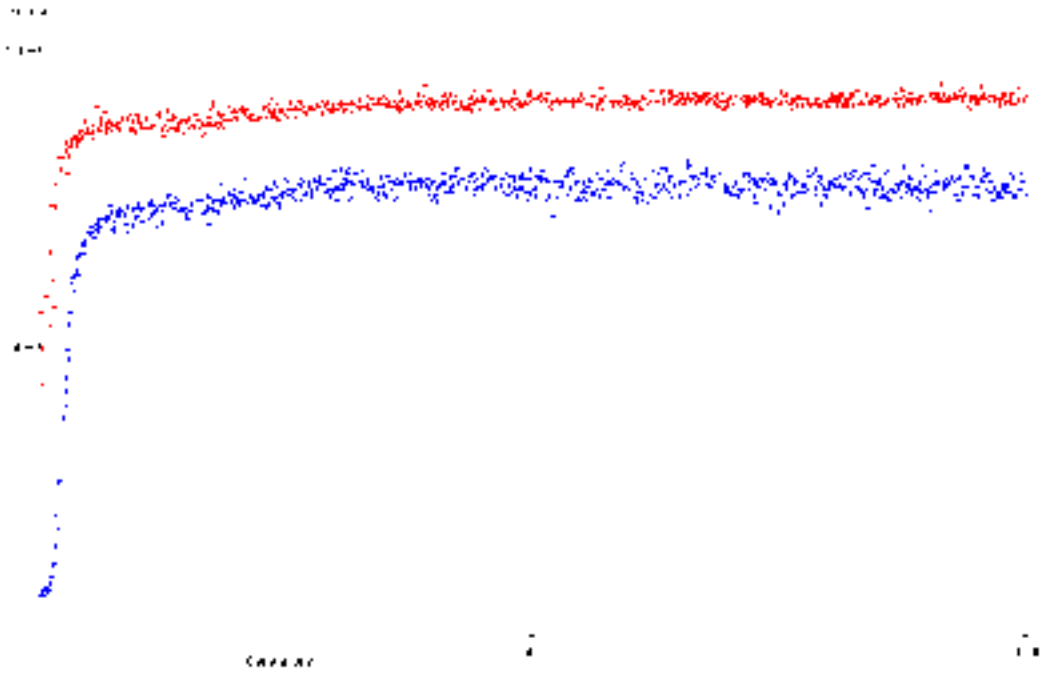


fig. 9 fitness results from a typical evolutionary run

Experiment 2

Again a population of 200 robots was evaluated for 1000 generations. In this task the robot was required to approach the light with no sensors after having been allowed to spin in place (with sensors) for 4 seconds. Two strategies emerged on separate runs. The first was simply to describe a large circle (*fig. 10*). As the light was always within a certain range of the robots starting position a robot that used this strategy stood a good chance of passing through the scoring zone and sometimes finishing near the light.

A second, more interesting strategy was displayed by a robot that would spin until it was facing the light and then stop. When released, it would travel a certain distance and then spiral sharply to the left, often within the scoring zone (*fig. 11*). In this case the robot always travelled in the direction of the light, but still seemed to rely on the law of averages to determine the distance of its spiral from the start position - the distance did not seem particularly to depend on the nearness of the light. Nevertheless, this delay before spiralling displays well the time-variant nature of the CTRNN.

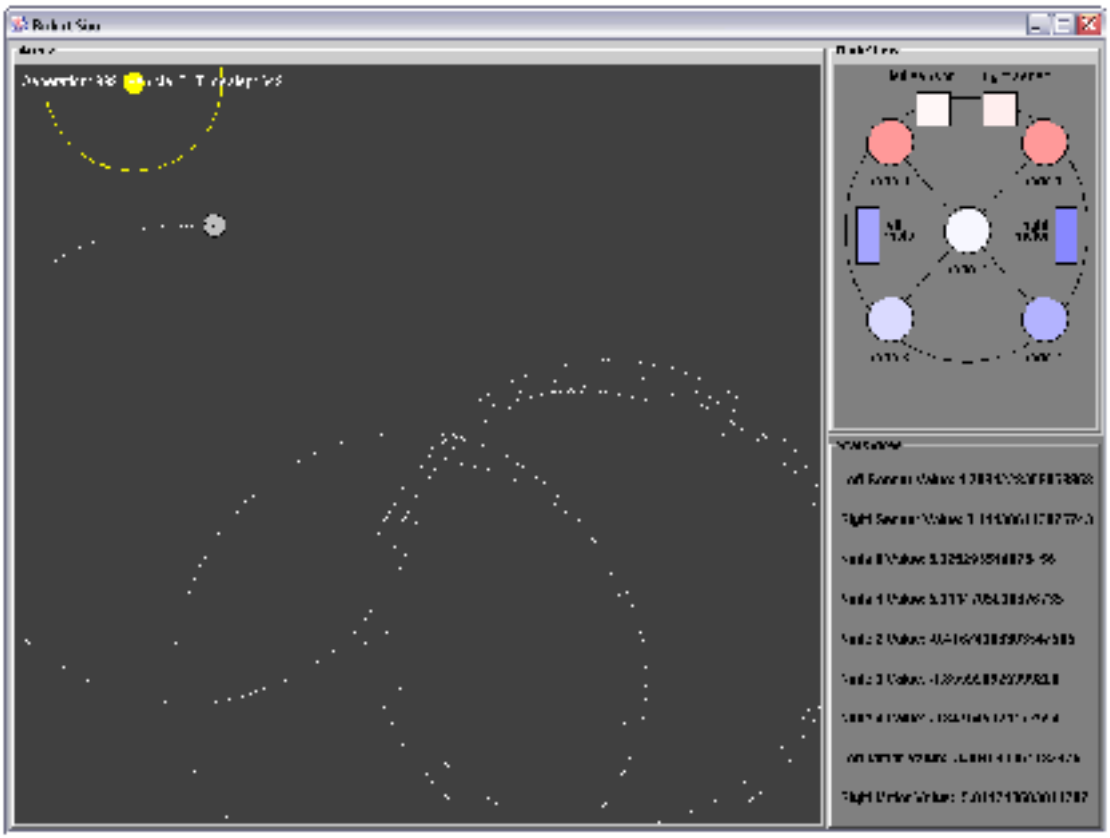


fig. 10 experiment 2 'circle' strategy

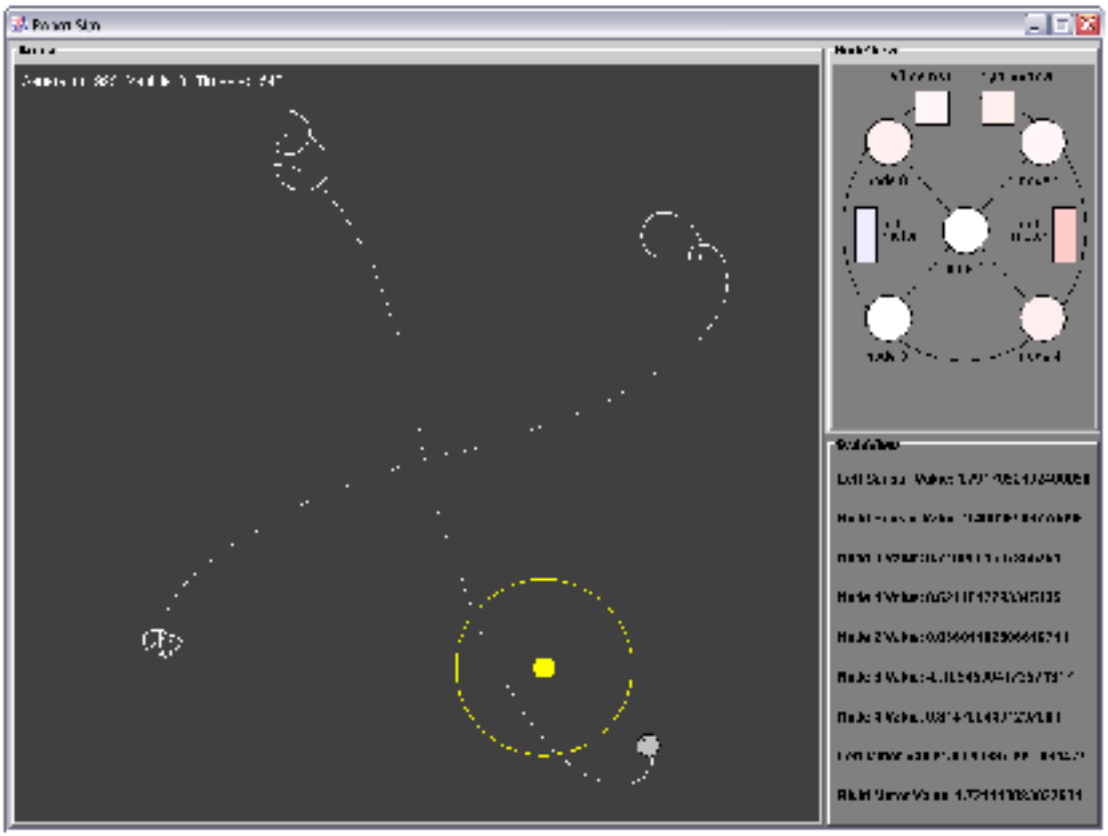


fig. 11 experiment 2 'spiral' strategy

Extensions

There are many further experiments that could be undertaken using this simulator, including altering the GA parameters, altering the weighting of the fitness function, adding more sensors, adding more light sources and so on. One interesting course would be to model the elements in the simulation more accurately (possibly including real world physics such as friction and mass) and attempt to transfer the controller to a real robot.

Conclusion

This report has described a successful evolutionary robotics project to evolve the behaviour of a population of robots to perform positive phototaxis. Although this task is simple it has demonstrated the principles of evolutionary robotics. A second experiment has displayed evolved behaviour caused by the time-variant nature of CTRNNs. From these results it is apparent that evolutionary simulation can be a powerful tool for investigating autonomous agents, providing useful data in a fraction of the time needed for experiments with real robots.

References

[1] Beer, R. (1995). *"On the dynamics of small continuous-time recurrent neural networks"*. Adaptive Behaviour 3(4):469-509.

[2] Braitenberg, V. (1984) *"Vehicles. Experiments in Synthetic Psychology"*. Cambridge, MA: MIT Press.

[3] Brooks R.A. (1991) "*Intelligence without reason"*. In Mylopoulos, J & Reiter, R (Eds.), Proceedings of 12th International Joint Conference on Artificial Intelligence. San Mateo, CA: Morgan Kaufmann.

[4] Cliff, D, Miller, G (1996), "*Co-Evolution of Pursuit and Evasion II: Simulation Methods and Results*". In Maes, P. et al. (Eds.), From Animals to Animats IV, Procs. of the Fourth Int. Conf. on Simulation of Adaptive Behaviour, pages 506-515, MIT Press.

[5] diPaulo, E. (2000). *"Homeostatic adaptation to inversion of the visual field and other sensorimotor disruptions"*. In Meyer, J-A, Berthoz, A, Floreano, D, Roitblatt, H and Wilson, S (Eds.), From animals to animats VI: proceedings of the 6th International Conference on Simulation of Adaptive Behaviour, pages 440-449. Cambridge, MA: MIT Press.

[6] Jakobi, N. (1998) "*Minimal Simulations for Evolutionary Robotics"*. PhD thesis, School of Cognitive and Computing Sciences, University of Sussex.

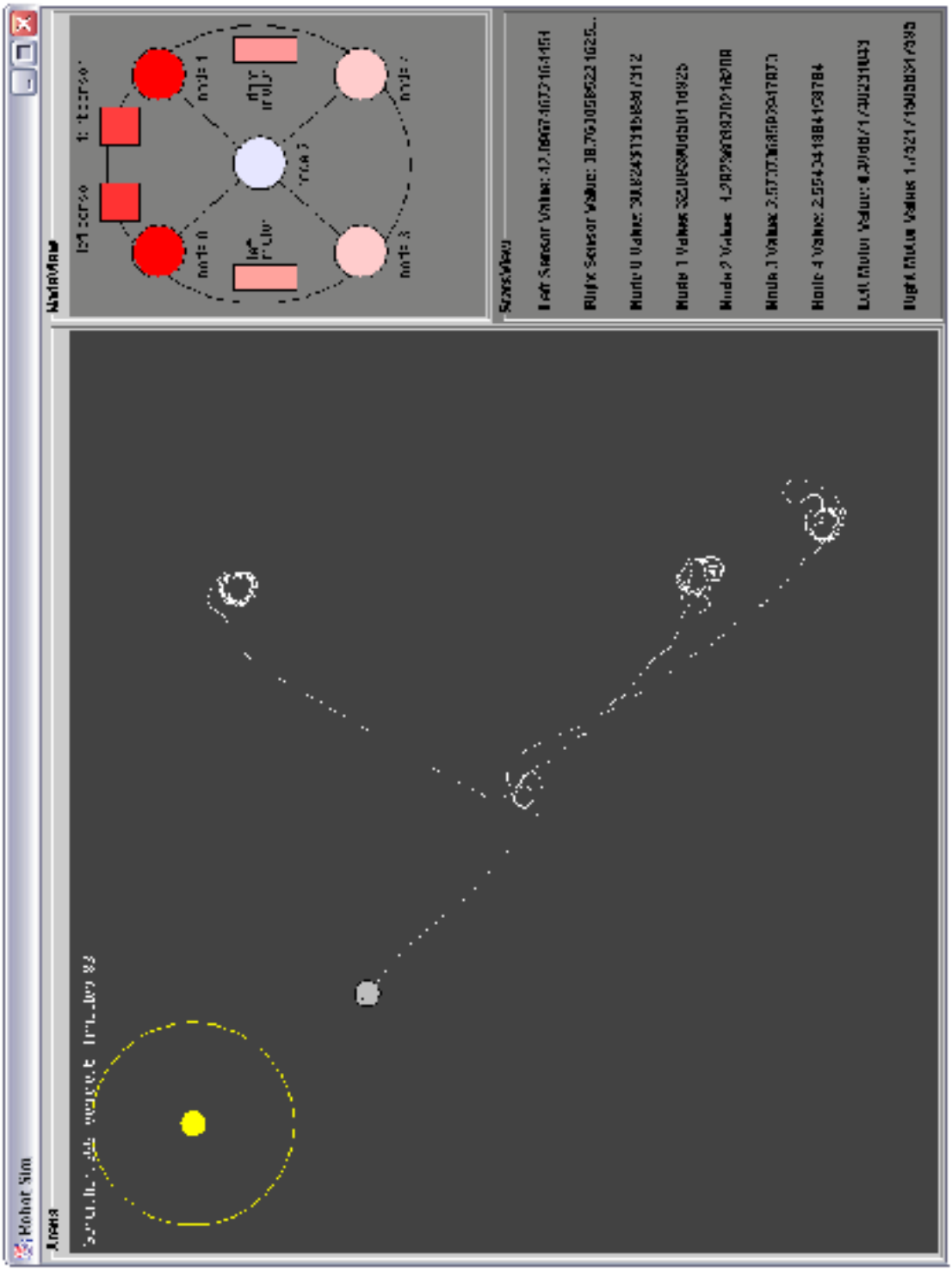
[7] Jakobi, N. (1997) "*Half-baked, ad hoc, and noisy: minimal simulations for evolutionary robotics*". In Husbands, P and Harvey, I, (Eds), Fourth European Conference on Artificial Life, pages 348-357. MIT Press.

[8] Sims, K. (1994). *"Evolving 3D Morphology and Behavior by Competition"*. In *Artificial Life IV*, Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, pages 28-39, MIT Press.

[9] Tuci, E, Harvey, I and Quinn, M. (2002*). "Evolving integrated controllers for autonomous learning robots using dynamic neural networks".* In Hallam, B, Floreano, D, Hallem, J, Hayes, G and Meyer, J-A, editors, *From Animals to Animats VII* Cambridge, MA, 4-9 August 2002. MIT press.

Appendix A

Colour slide of simulator screen



Appendix B

The Code

Java 1.4.2

RobotSim.java
The main runnable class, controls the evaluation and GA

Vehicle.java
The robot model

Genome.java
A collection of NodeDescriptors, the blueprint for the CTRNN controller of the robot

NodeDescriptor.java
Holds information about one of the nodes in the CTRNN

Network.java
A model of the CTRNN network, performs the network calculations

Node.java
A model of a single node in the network

Arena.java
GUI class, the simulator's arena window

ArenaCanvas.java
GUI class, the canvas on which the simulation is drawn

StatsView.java
GUI class, displays numerical data

NodeView.java
GUI class, displays the state of the network

NodeViewCanvas.java
GUI class, the canvas on which the node view is rendered