**Tristan Fletcher** **20/03/2006**

# Further Analysis in the Evolution of Dynamical Neural Networks for Minimally Cognitive Behaviour

## Table of Contents

# 1. Introduction

## *1.1 Approach*

The Situated, Embodied and Dynamical approach to cognition (SED) (Beer 1997) emphasises the role of the agent's physical body, its environment and its brain in the generation of its behaviour. By making many simplifying assumptions concerning the characteristics of these three components, it is possible to explore the behaviour of simple agents and relate it to these characteristics. A dynamical perspective can be taken so that the concepts and mathematical tools of dynamical systems theory are applied to the analysis of these complex (in a literal, mathematical sense) systems. This combined methodology of adopting the SED perspective, making simplifying assumptions and using dynamical systems theory facilitates the answering of questions relating to adaptive behaviour. An example of such questions might be how an agent's neuronal architecture affects its behaviour, what influence the size of its environment has and how contingent its behaviour is on its body shape.

The work of Beer (1996 & 2000) is initially followed very closely for evolving agents that can track objects and then discriminate between them. A template for a biologically feasible agent with an analytically tractable neuronal architecture is created and then evolved in a very similar manner as to Beer's experiments. However, once this has been done, the behaviour of these agents is scrutinised from a more behavioural and no-linear perspective and hence represents a departure in terms of the focus of the investigation from Beer's work.

## *1.2 Overview of Experiments*

Experiments were first conducted to evolve simple agents which would track objects falling from a range of horizontal offsets to the agent. The generalised performance of these agents in tracking objects dropped from offsets not used in the agents evolution was then investigated. A more sophisticated agent was then evolved which could discriminate between two different classes of object – tracking one kind and avoiding the other. The agent's generalisation was also investigated for the discrimination task. The behaviour of two agents evolved to carry out the previous task embodied in combinations of the two object types was then investigated.

## *1.3 Hypotheses*

It is believed that the generalisation ability of simple agents to track objects will be good for objects that are dropped within the ranges of offsets used in the agents' evolution but poor outside these ranges. This is also hypothesised to be true for the more complex object discrimination agents.

At a higher level, it is believed that the neuronal response of an agent evolved to discriminate between classes of objects would be very different depending on the class of object being perceived. Furthermore, it is thought that the manner within which the agent moves in order to track objects will be very different to the way it moves when avoiding them. For this reason, it is hard to envisage the emergent behaviour of dual agent systems, though it is believed that their behaviour will be highly non-linear.

## 2. Methods

In the experiments that follow, the agent exists in an environment of size 400 × 275. It has a circular body with a diameter of 30 with an eye consisting of 7 rays of maximum length 220, uniformly distributed over a visual angle of $\pi/6$[1]. An intersection between a ray and a neuron causes an input to be injected into the corresponding sensory neuron, the magnitude of which is inversely proportional to the distance to the object. These inputs range between 10 for rays that are of zero length to 0 for rays at their maximum length.

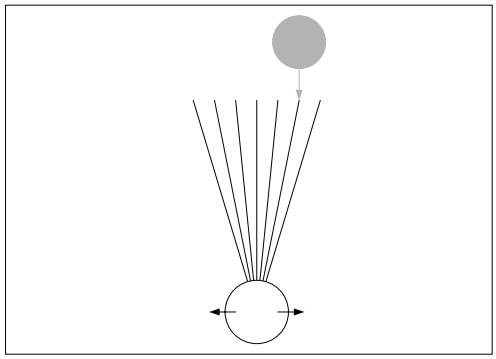


***Figure 1:*** *An agent with rays on the point of perceiving an object in its environment.*

The agent's neural architecture consists of 7 ray sensor neurons projecting to 2 motor neurons controlling horizontal motion.[2] The neurons are networked in a continuous-time recurrent neural network (CTRNN) with the following state equations:

$$\frac{\partial y}{\partial t}_i = \frac{1}{\tau_i}(-y_i + I_i) \qquad \text{for the sensory neurons } (i = 1,\ldots 7)$$

$$\frac{\partial y}{\partial t}_i = \frac{1}{\tau_i}\left[-y_i + \sum_{j=1}^{7} w_{ij}\sigma(g_j(y_j + \theta_j))\right] \qquad \text{for the motor neurons } (i = 1, 2)$$

$$y_i = \sigma(g_j(y_j+\theta_j)) \qquad \text{for the outputs to the motors } (i = 1, 2)$$

where $y_i$ is the state of the $i^{\text{th}}$ neuron; $\tau_i$ is its time constant; $w_{ji}$ is the strength of the connection from unit $j$ to unit $i$; $\sigma(x)$ is the standard logistic activation function:

$$\sigma(x) = \frac{1}{1 + e^{-g(x+\theta)}}$$

with bias term $\theta$, gain $g$ and $I_i$ representing an external (sensory) input. States were initialised to 0 and circuits were integrated using the forward Euler method with an integration step size of 0.1. Time constants $\tau_i$ of 1 were used in all the experiments.

[1] The actual units that these measurements are in are arbitrary.

[2] Such a simple neuronal architecture was adopted at this point, to remain consistency with Beer's paradigm of maintaining the simplest possible agent-environment system that raises issues of genuine cognitive interest.

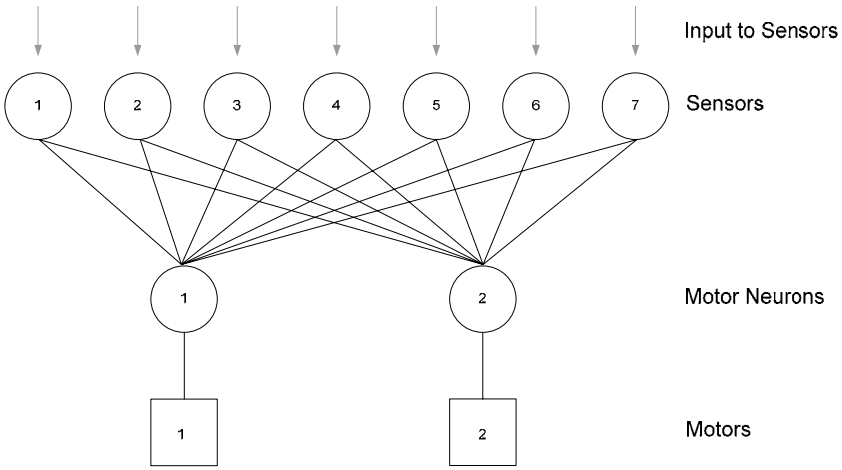



***Figure 2:*** *Neural architecture for initial orientation experiment*

Matlab's Genetic Algorithm toolbox was used to evolve the agents where described. However, the toolbox does not supply a bounded Gaussian mutation function, so it was necessary to write one: see *ConstrainedGaussianMutation.m* in appendix B. A population of individuals was maintained, with each individual encoded as a vector of real numbers representing the connection weights $w_{ji}$, the biases $\theta_i$ and the time constants $\tau_i$, as well as the gains $g_i$ of the ray sensory neurons, with all other gains fixed to 1. Random populations of individuals were generated by initialising every component in each individual to random values distributed over the range ±1. Individuals were selected for reproduction using fitness proportional selection (`selectionroulette` in Matlab) with linear fitness scaling (`fitscalingprop`). A selected parent was mutated by adding it to a random displacement vector whose direction was uniformly distributed in every dimension and whose magnitude was a Gaussian random variable with 0 mean and a variance of $\sigma^2$. For each slot in the new population, the child was chosen if its performance was greater than or equal to that of the parent, otherwise the parent was copied. All random numbers were generated using the routine `randn` which has a period $\approx 2^{64}$.

# 3. Agent Evolution Experiments

## *3.1 Simple Orientation*

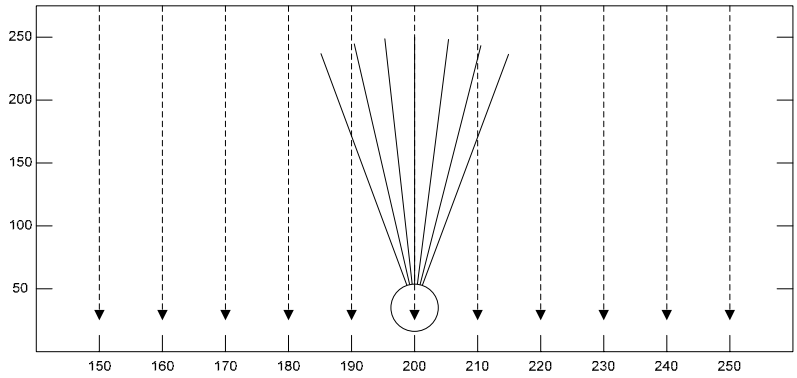



***Figure 3:*** *The 11 fitness evaluation trials for the simple orientation task*

In the first set of experiments to be described, agents were evolved that could use their vision to adjust their horizontal position so as to catch falling objects. Their horizontal velocity was proportional to the sum of the opposing horizontal forces produced by each motor (with a constant of proportionality of 5). Circular objects with a diameter of 26 were dropped from the top of the environment with horizontal offsets of ±50, ±40, ±30, ±20, ±10 and 0 from the centre of the agent. These objects had no horizontal velocity but had a vertical velocity of 10. See *OrientationEvolve.m* in appendix B.

The performance measure to be minimized was:

$$\sum_{i=1}^{NumTrials} d_i \Big/ NumTrials$$

Where *NumTrials* is the total number of trials (11 in this case) and $d_i$ is the horizontal distance between the centres of the object and the agent when their vertical separation goes to 0 on the $i^{th}$ trial.

10 parameters were used in the evolution of an agent to solve this tracking task. The weights between the sensory neurons and the motor neurons were arranged in a bilaterally symmetric fashion, requiring only 7 parameters. The other 3 parameters were for the gain and the bias of the sensors and the bias of the motors (with the gain of the motors set to 1). The parameter ranges were as follows: connection weights $\in$ [-6,6], biases $\in$ [-10,10] and gains $\in$ [-10,10].

Populations of 25 individuals were evolved for 200 generations with a mutation variance $\sigma^2$ of 1 with a resulting fitness of 97.25%[3] for the best agent - see the evolution graph in Appendix A.

### *3.2 Object Discrimination*

In an attempt to investigate whether the agent could discriminate between circular objects and lines, another experiment was conducted. In a similar manner as to before, objects were dropped from different horizontal offsets to the agent (±40, ±30, ±20, ±10 and 0) and the agent was evolved to track the object if it was a circle and move as far away from it as possible if it was a line. In order to make this discrimination task slightly easier for the agent, the vertical velocity of the objects was reduced to 5 and the circle's diameter increased to 30.

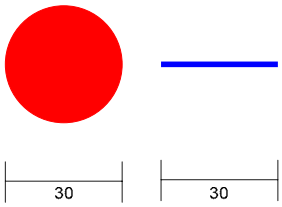



***Figure 4:*** *The two classes of object used in the object discrimination task*

[3] Throughout this report, fitness will be reported as a percentage of the performance of the agent if it had remained immobile.

The performance measure was adapted so that the sum of the horizontal distance of agents from circular objects was minimised whilst the distance of agents to lines was maximised. However, this latter component of the measure was capped at 50 to prevent the avoidance of lines by large distances from dominating the fitness at the expense of accuracy in catching circles.

In order for the agent to carry out this discriminatory behaviour, it was necessary for it to possess an intermediate layer of 5 neurons. In terms of the CTRNN state equations earlier, the extra layer would be described by:

$$\frac{\partial y}{\partial t}_i = \frac{1}{\tau_i}\left[-y_i + \sum_{j=1}^{7} w_{ij}\sigma(g_j(y_j + \theta_j))\right] \quad \text{for the inter-neurons } (i = 1,..5)$$

And the equation for the motor neurons would change slightly to:

$$\frac{\partial y}{\partial t}_i = \frac{1}{\tau_i}\left[-y_i + \sum_{j=1}^{5} w_{ij}\sigma(g_j(y_j + \theta_j))\right] \quad \text{for the motor neurons } (i = 1, 2)$$

As before, the sensors shared the same bias and gain. However, each of the inter-neurons had their own bias and gain whilst the weights from the sensors to these inter-neurons were arranged in a bilaterally symmetric manner. The connections between the inter and motor neurons were not arranged in a bilaterally symmetric fashion however. This was because the author felt that though this would mean that the agent's evolution might be more complex due to the increased number of parameters, a fully bilaterally symmetric nervous system would be unnecessarily restrictive and removing this constraint could produce more interesting behaviour. The total number of parameters being altered in the agent's evolution was therefore 45. See *ObjectDiscriminateInterNeuronsEvolve.m* in Appendix B.

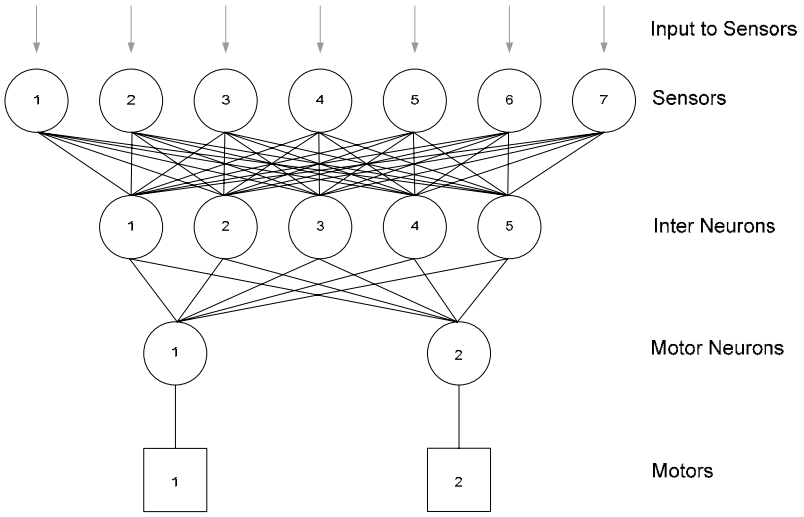



***Figure 5:*** *Neural architecture for object discrimination experiment*

Populations of 200 individuals were evolved for 2000 generations with a mutation variance $\sigma^2$ of 10 with a resulting fitness of 99.64% for the best agent (the evolution graph for this can not be shown as the process was carried out in several stages).

# 4. Generalisation Experiments

The broad concept of the generalisation experiments was to see how agents evolved to perform certain tasks would perform in conditions that had not been present in their evolutionary 'training'.

### *4.1 Simple Tracking Spatial Generalisation*

For the relatively simple agent that was evolved to track circles, the performance of the agent was evaluated for initial object horizontal offsets that were not used in the evolutionary process's fitness function. A range of horizontal offsets between ± 80 in increments of 2 was investigated.

### *4.2 Object Discrimination Spatial Generalisation*

The previous experiment was repeated for the comparatively complex agent that had been evolved to discriminate between circles and lines. The same set of horizontal offsets was used, and the behaviour for both circles and lines investigated.

# 5. Dual Agent System

The behaviour of agents was then investigated in systems where the object that the agent was discriminating and then either tracking or avoiding was replaced by another agent. This other agent was approaching the first agent with a constant vertical velocity as before, though this time with a lower value of 2, from a range of initial horizontal offsets. The agents used were the best evolved agents from the object discrimination task. Investigations were carried out for systems where both agents were circles, both were represented as lines and where one was a line and the other a circle. See *DualAgentSystem.m* in Appendix B.

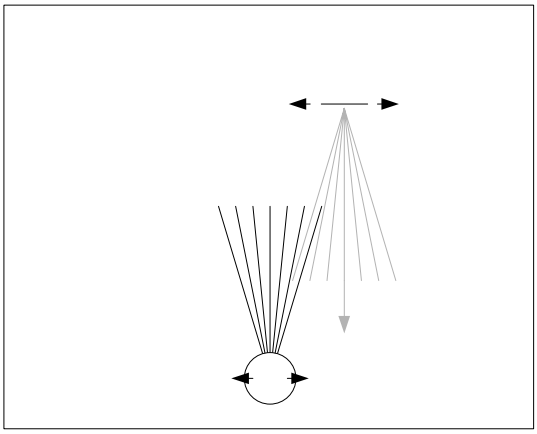


***Figure 6:*** *A tracking agent approaching an avoiding agent in a Dual Agent System*

# 6. Performance Behaviour

## *6.1 Simple Orientation*

The best evolved agent was able to track the circular objects which were falling towards it from the initial horizontal dropping offsets used in its evolution. The horizontal distance between the agent and the object when the latter's vertical displacement was zero was low for all of the trials as Figure 7 shows. Furthermore, the performance of the agent for horizontal offsets not used in its evolution, i.e. its generalised performance, was good for the range of offsets used in evolution (between ± 50) but poor outside this range as Figure 8 shows.

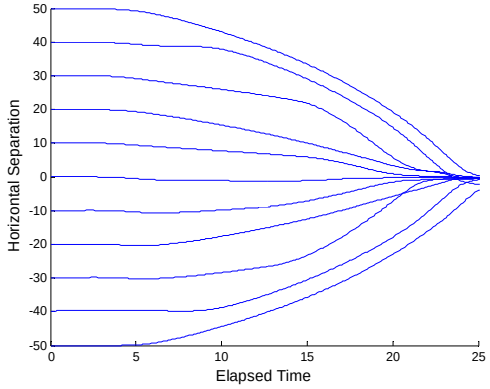



***Figure 7:*** *Horizontal separation between object and agent over time*

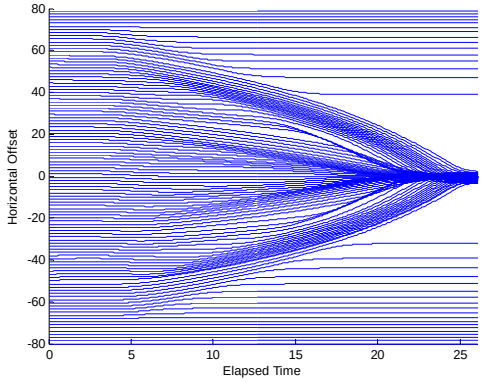



***Figure 8:*** *Generalised performance of simple tracking agent over time*

## *6.2 Object Discrimination*

The best evolved object discriminating agent was able to track circles and avoid lines which had been dropped from the offsets used in its trials as Figures 9 and 10 show.

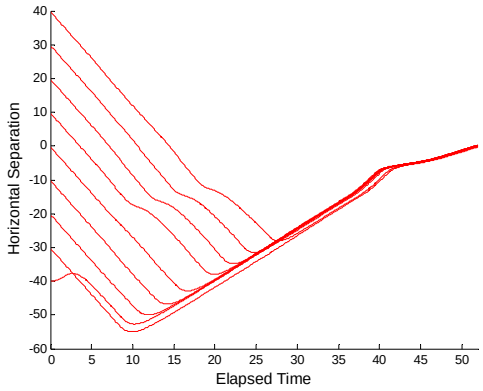



***Figure 9:*** *Horizontal separation for circles over time*

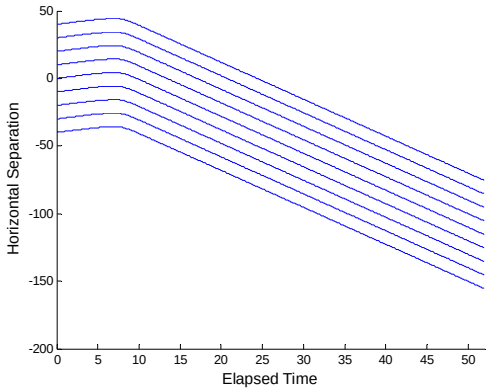



***Figure 10:*** *Horizontal separation for lines over time*

The performance of this agent in tracking circles and avoiding lines for horizontal offsets not used in its evolution, i.e. its generalised performance, was good for offsets > -40 but poor outside this range for circle tracking. However, for line avoiding the agent's performance was good for all offsets.

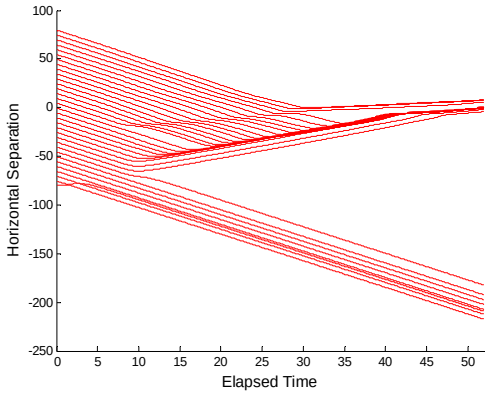



***Figure 11:*** *Generalised performance of the agent tracking circles*

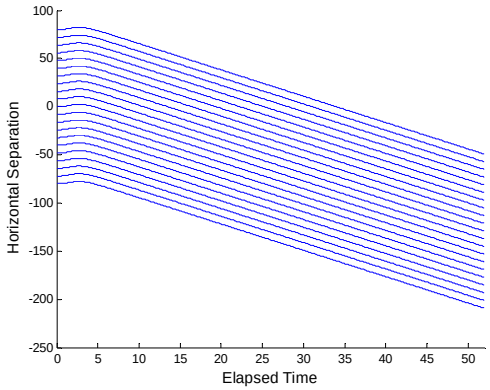



***Figure 12:*** *Generalised performance of the agent avoiding lines*

The horizontal position of the agent over time when it is approached by a circle, line and nothing at all is shown in Figure 13. For the sake of completeness, the final separation between agents tracking circles as a function of their initial horizontal offset is shown in Figure 14.

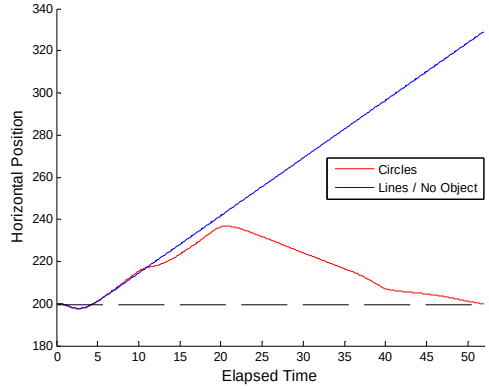



***Figure 13:*** *Horizontal position of agent when approached by different objects*

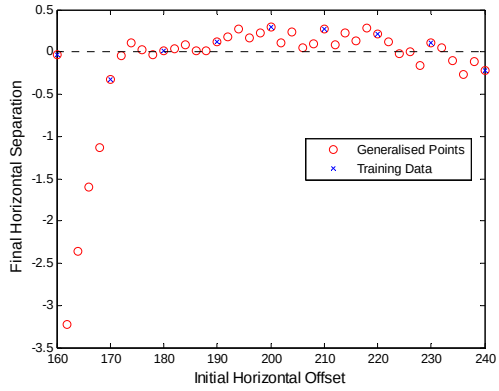



***Figure 14:*** *Final separation between agent and circle for a range of initial offsets*

The agent's active responses to objects approaching it can be characterised by examining its reaction to objects held at fixed distances as Figures 15 and 16 evince. These figures show the agent's horizontal offset after 50 time units from objects held stationary in a range of vertical and horizontal positions.

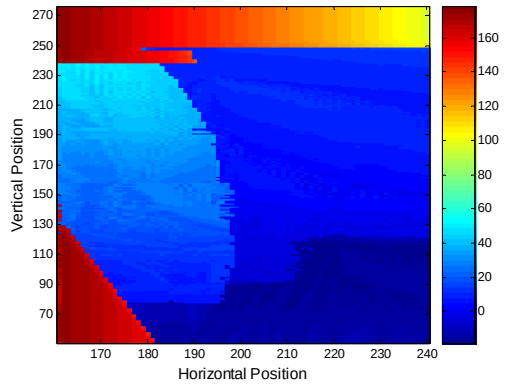



***Figure 15:*** *Final horizontal offset of agent for circles held at different fixed positions*

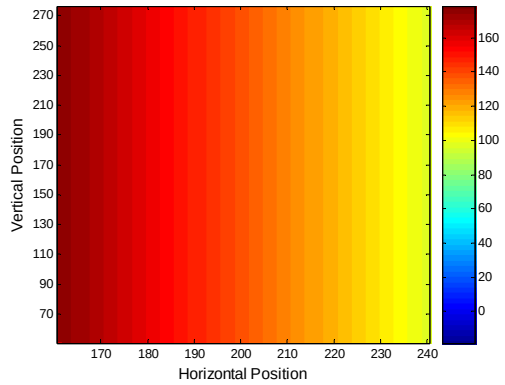



***Figure 16:*** *Final horizontal offset of agent for lines held at different fixed positions*

The figures show that the agent has exactly the same behaviour for lines as it does for no object at all: moving to the right[4] after an initial short movement to the left. The behaviour for circles is slightly more complex: the agent moves away from objects held at large vertical positions and those held at both low vertical and horizontal positions but tracks all other objects reasonably well – performing slightly better with objects to its right.

### *6.3 Dual Agent System*

The horizontal separation of a system where both agents are circles and hence attracted to each other is shown as a function of the agents' vertical separation in Figure 17 for different initial horizontal offsets. The actual horizontal positions of both agents as a function of their vertical separation is shown in Figure 18. These figures are repeated for systems where one agent is a line and the other a circle (Figures 19 & 20) and where both are lines (Figures 21 & 22).

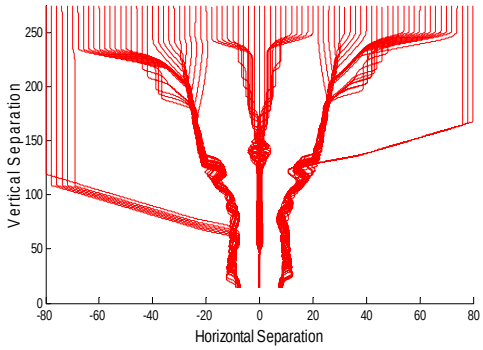



***Figure 17:*** *The horizontal separation of two circular agents as a function of their vertical separation*

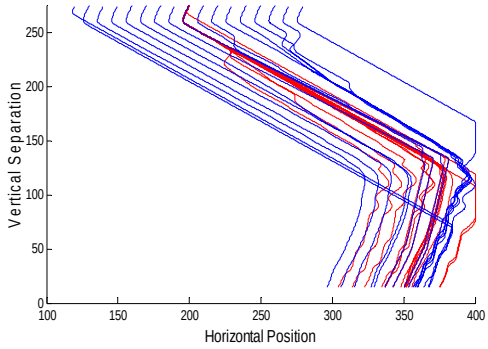



***Figure 18:*** *The actual horizontal position of two circular agents as a function of their vertical separation*

For agents which are attracted to each other, it appears that for initial horizontal offsets of ± 20, the agents end up with a minimal separation. However, for initial offsets < -20 the final separation is ≈ -10 and for offsets > +20 it is ≈ +10. This can be seen in Figure 18 by the red lines oscillating around their appropriate blue line with little offset in some cases and a greater offset in others. Furthermore, there is a reasonable range of final horizontal positions for both agents.

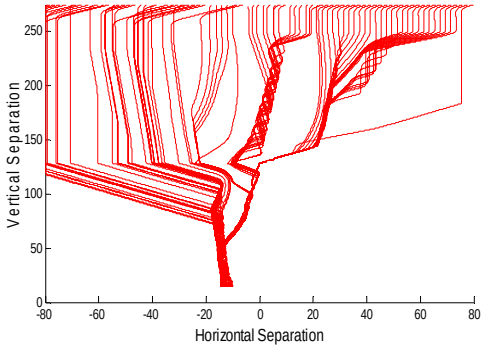



***Figure 19:*** *The horizontal separation of a circular agent with a line agent*

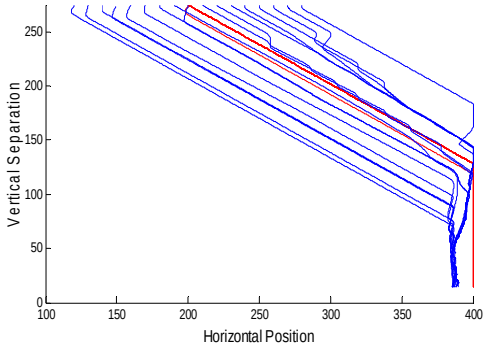



***Figure 20:*** *The horizontal position of a circular agent (red) and a line agent (blue)*

[4]Throughout this report, the term 'left' indicates a horizontal position of relatively smaller magnitude and 'right' one of relatively greater magnitude.

In a system with a circular agent and a line agent, the final separation is ≈ -10 for all initial horizontal offsets. However, for initial offsets < -20 the final separation is ≈ -10 and for offsets > +20 it is ≈ +10. Figure 18 shows that there is a very narrow range of final horizontal positions for the line agents at about 380 and the circular agents at 400.

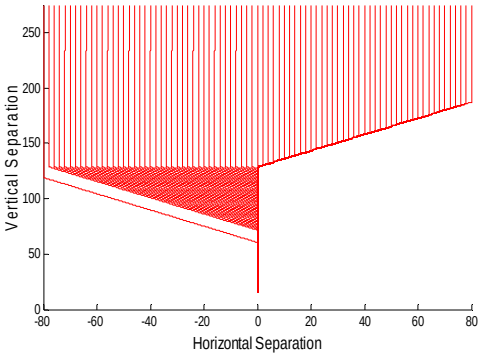



***Figure 21:*** *The horizontal separation of two line agents*

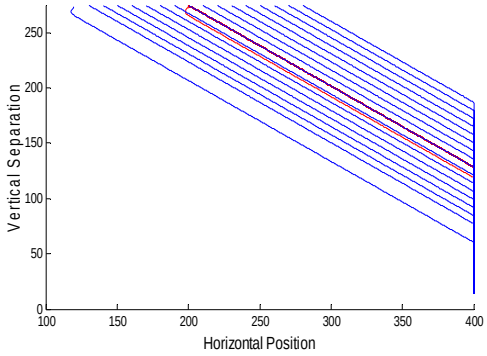



***Figure 22:*** *The actual horizontal position of two line agents*

These figures show that where both agents are lines and hence avoiding each other, the final horizontal position is always 400 for both agents and therefore their separation is always 0.

# 7. Explanation of Results

## *7.1 Dynamics*

### 7.1.1 Simple Orientation

Referring back to Figure 8, the simple tracking agent can successfully track circles falling from within the horizontal ranges that it was trained to do so (i.e. ±50). For objects that fall just slightly outside this range the agent initially tracks them (though less vigorously than objects with initial offsets within the range) and then stops as the object has fallen out of view of the agent's sensor rays. Objects falling significantly out of the range are never perceived by the agent. This is because the agent's default velocity is zero and it only moves once it has perceived an object in its field of vision.

### 7.1.2 Object Discrimination

The performance of the more complex discriminating agent is best understood by first referring to Figure 13. The movement of the agent when there is no object falling shows that the agent's default motion is to move to the right. The agent makes no change to its motion when it perceives a line and only changes its direction when it has perceived a circle.

Figure 9 shows that when agents perceive circles, they first actually move away from them in their default movement direction but at approximately twice the default rate. When the circles have reached a certain level of vertical proximity the agent then switches the direction of its movement so that by the time their vertical displacement has reached zero, the agent has 'caught' the object.

The shape of the graph in Figure 11 can now be understood: the agent is able to track circles dropped to the right of the initial offset it was trained for (i.e. > +40) because its default movement is in this direction anyway. So though the agent may take slightly longer to perceive the object, as it sweeps to the right it will eventually do so. However, objects dropped slightly to the left of the training range (i.e. < -40 & > -50) are perceived successfully but the agent waits longer before it does its sudden movement switch with these objects and hence they are caught with slightly lower success. Objects dropped from initial horizontal offsets < -50 are never noticed by the agent and therefore never tracked.

For lines, in all cases the agent simply moves towards the right with an initial movement to the left.

### 7.1.3 Dual Agent System

The more complex behaviour of agents in the dual agent system appears to be more difficult to explain. In a system where both agents are tracking to each other there appear to be three basins of attraction or stable end states of the system – shown by the three main stems in Figure 17.

Looking at the left and right stems, i.e. for systems where one agent is approaching at an offset of > +25 or < -25, both agents initially move to the right. For the stationary

agent this is because this is its default movement, for the 'dropped' agent this is because it has perceived the other agent and is tracking it. When the agents get slightly closer, they both suddenly switch to moving to the left. However, the agents never actually 'catch' each other as their tracking velocities appear to always over-compensate as they attempt to meet each other; this is seen in the swirling patterns at the bottom of Figures 17 and 18.

The middle stem represents agents that are able to perceive each other more effectively as there is a smaller horizontal offset between them. This means that both agents track each other effectively and have already 'caught' each other before their vertical separation has dropped below the distance required for their movement to suddenly switch from right to left.

In systems where one agent is seeking but the other avoiding, the behaviour is even more simple to explain. Both agents initially move in their default direction to the right. However, it takes longer for the sudden switch in the direction of the tracking agent to occur, by which point the avoiding agent has already been restricted by the horizontal dimensions of their environment and had been remaining at a horizontal position of 400. This means that the avoiding agent had not had significant opportunity to move away from the tracking agent before their vertical separation was 0 – hence the single stem of a small final displacement.

In systems where both agents are avoiding each other, they both move to the right. This has the counterintuitive result that their final separation is always zero. The reason for this is that both agents are restricted by the dimensions of the environment and have horizontal positions of 400 by the time their vertical displacement is zero.

### *7.2 Nervous System*

The behaviour of the agent as explained above can now be understood in terms of its neuronal behaviour by looking at the activity of its neurons over time. Figures 23 to 30 show the outputs of each layer of neurons for the best evolved agent when perceiving an object falling from an offset of -40 for both circles and lines.

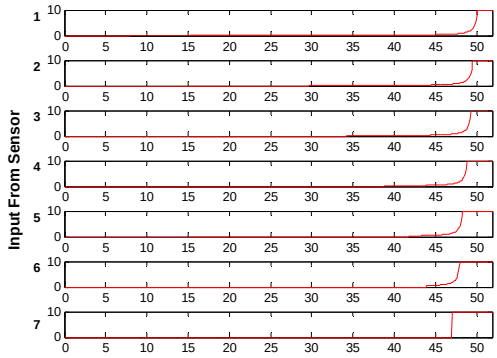



***Figure 23**: Input from Sensors (Circle)*

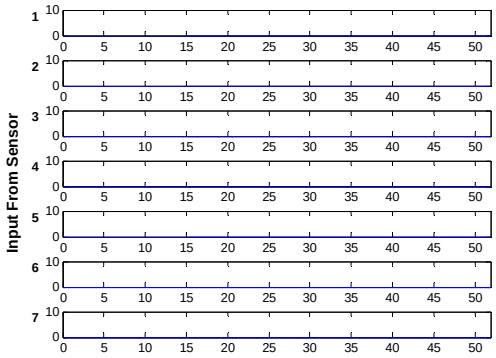



***Figure 24**: Input from Sensors (Line)*

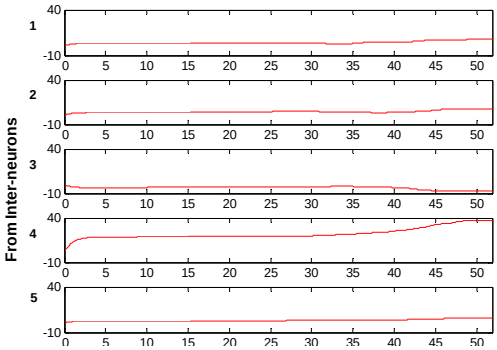



***Figure 25**: Output from INs (Circle)*

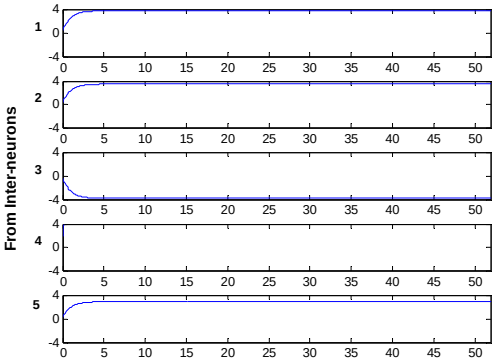



***Figure 26**: Output from INs (Line)*

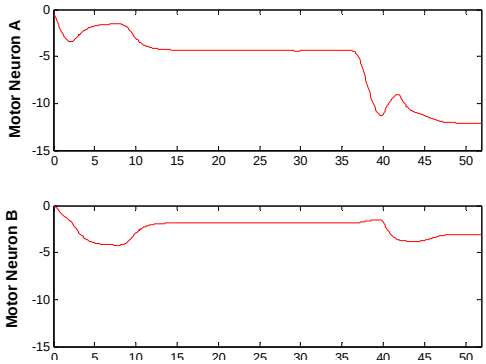



***Figure 27**: Output from MNs (Circle)*

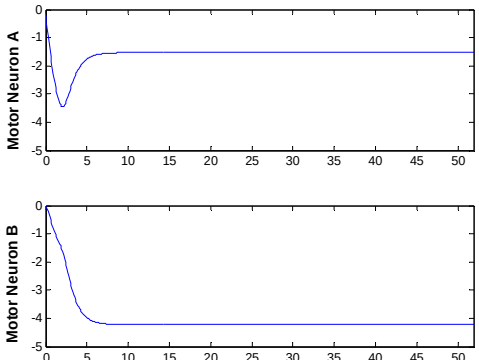



***Figure 28**: Output from MNs (Line)*

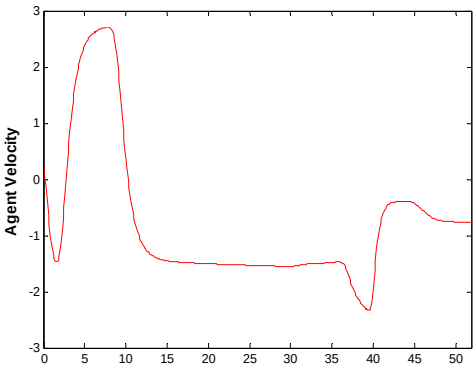



***Figure 29**: Agent Velocity (Circle)*

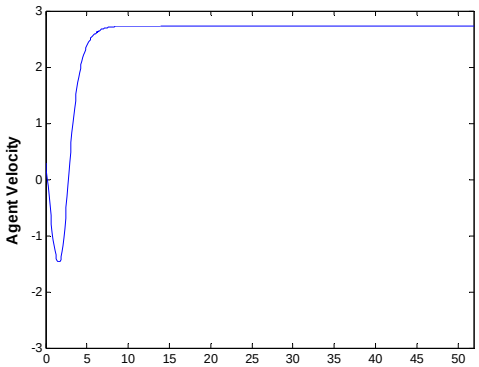



***Figure 30**: Agent Velocity (Line)*

**Elapsed Time**

**Elapsed Time**

If one compares Figures 23 and 24, one can see that the first sensor (the left-most one) foveates the circle after about 7 time units, with the other sensors gradually following suit and then all sensors suddenly move towards saturation after 45 time units. For the agent perceiving a line, the neuronal response is completely different. The bias and gain of these neurons is clearly such that these sensors never output any values – even when the agent and line have a very small vertical separation.

This difference between the agent's sensor response to a circle and line means that for the figures relating to the agent's response to a circle, the neuronal responses can be understood by the signals from the sensors being propagated down through the different neuronal layers. However, for the figures depicting the agent's response to a line, this is not the case. The neuronal responses are the default behaviour of the agent when there is no output from the sensors. The behaviour of all the neurons for the case where an agent perceives a circle and the case where it perceives a line is identical for the first 7 time units (though this may not be immediately apparent from the different scales the two sets of figures use).

Looking at the default behaviour of the agent, one can see that the inter-neurons quickly reach either positive or negative saturation. This means that after saturation has occurred at about 5 time units, the outputs from the motor neurons are constant – producing the default rightwards movement (at a constant velocity) of the agent. The period before this saturation of the inter-neurons causes a slight dip in Motor Neuron A's output, resulting in the initial leftwards movement that both line and circle agents exhibit in all cases.

The inter-neurons for the circle-perceiving agent do not reach saturation because they are clearly being inhibited by the non-zero outputs from the sensor neurons. Their combined effect on the motor neurons is more complex and results in the agent swinging back left after the default movement which it carries out in the first 5 time units. Though the agents velocity changes when it gets even closer to the circle, it still remains moving leftwards.

It therefore appears that from scrutinising the response of the agent to objects fixed at a particular position for circles and lines as see in Figures 15 and 16, one can see a marked difference in behaviour. Coupled with an investigation of the neuronal activity of the agent to these two different objects, one can see that there is a static difference in the dynamical behaviour of the agent to lines and circles. The agent appears to be statically classifying the objects it sees as either lines or circles – it is not employing a more sophisticated strategy such as active scanning.

# 10. Adaptation

## *10.1 Noise Perturbations*

The robustness of the agent's performance in catching circles and lines when subjected to varying degrees of sensorimotor noise was investigated. Gaussian noise with a zero mean but with increasing standard deviation was applied to the sensor and motor neurons[5]. The agent's performance was measured 10 times and the average plotted for a range of horizontal object-dropping offsets and standard deviations of noise.

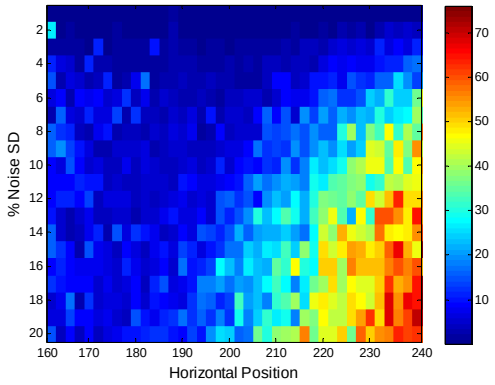



***Figure 31:*** *Final separation of circle tracking agent from circle for varying levels of sensor noise*

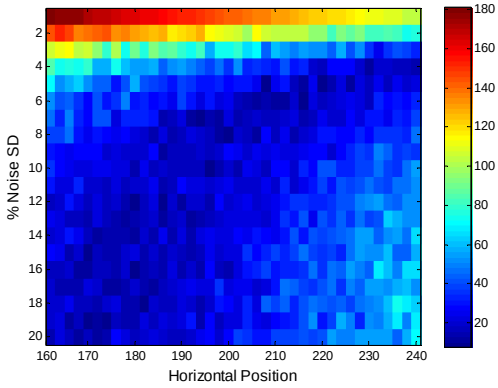



***Figure 32:*** *Final separation of line tracking agent from line for varying levels of sensor noise*

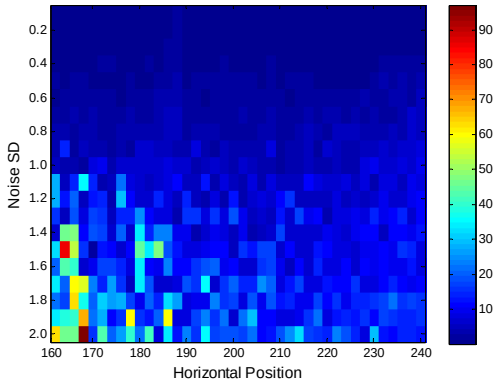



***Figure 33:*** *Final separation of circle tracking agent from circle for varying levels of motor neuron noise*

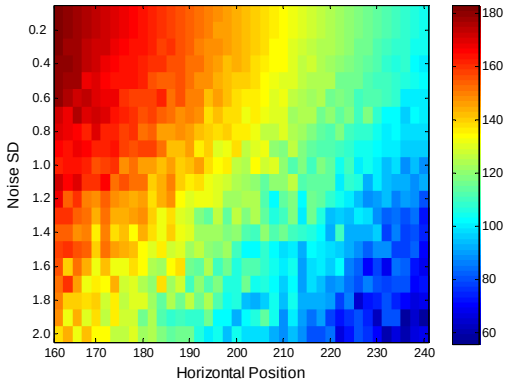



***Figure 34:*** *Final separation of line tracking agent from line for varying levels of motor neuron noise*

The circle tracking agent appears to be fairly insensitive to low noise levels at its sensor and motor neurons for all ranges of horizontal offsets. However, for more significant noise levels, the agent is much less effective at catching circles when the noise applied to its sensor neurons when the circle is approaching to the right of the agent and at the opposite side for noise applied at its motor neurons. When tracking

[5] When adding noise to the sensor neurons, the standard deviation of the noise was a percentage of the maximum sensor input of 10. For the sake of comparison, the same scale of standard deviations was used for adding noise to the motor neurons – though it would no longer be correct to state this as a percentage of anything.

circles, the agent is more sensitive to noise applied to its sensor neurons than it is by noise applied to its motor neurons.

It is important to remember that the graphs showing the line catching agent's sensitivity to noise are showing final separation, so colours detailing high values here signify better performance (as opposed to the graphs describing the circle tracking agent's performance). The line catching agent's performance is significantly more sensitive than the circle catching agent when noise is applied to its sensor neurons. Low levels of noise of up to about 6% do not hinder the agent's performance as shown by the fact that the final separation is above 50. However, as the noise is increased above this level the agent's performance decreases markedly. The agent's performance in tracking lines appears to be unhindered by noise applied to its motor neurons for all horizontal offsets.

### *10.2 Lesions*

The agent's performance was investigated in cases when a sensor, inter or motor neuron was removed and also in situations where a connection between the sensor and inter neurons and the inter and motor neurons was severed.[6]

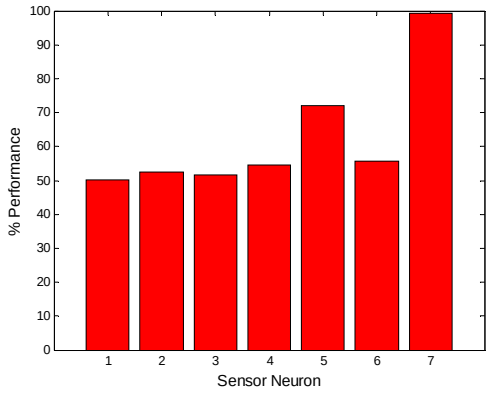



***Figure 35:*** *Sensitivity of agent performance to sensor neuron removal*

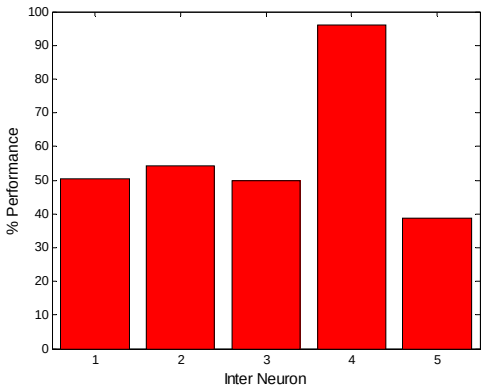



***Figure 36:*** *Sensitivity of agent performance to inter neuron removal*

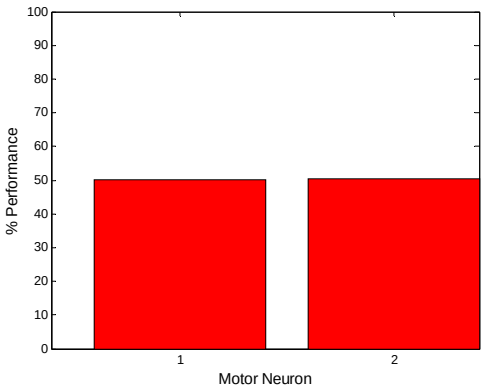



***Figure 37:*** *Sensitivity of agent performance to motor neuron removal*

[6] Percentage values here indicate the performance of the agent averaged over the range of horizontal offsets for both circle tracking and line avoidance as a proportion of the performance of a fully intact agent.

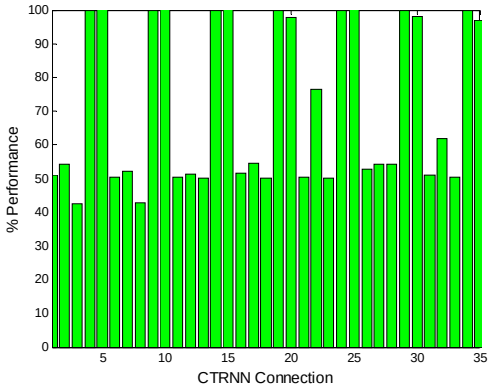



***Figure 38:*** *Sensitivity of agent performance to connection lesions between sensor and inter neurons*

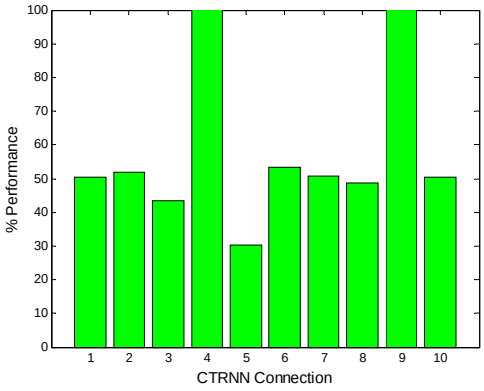



***Figure 39:*** *Sensitivity of agent performance to connection lesions between inter and motor neurons*

These figures indicate that the agent's performance would be hindered by the removal of most of the neurons but that removing the 7th (rightmost) sensor neuron or the 4th inter neuron would have almost no effect on the agent's performance. Furthermore, severing the connections between these neurons and other ones in the agent's brain has no effect on the agent's performance.

# 11. Discussion

Clearly, the restriction of the size of the agent's environment has had an impact on its ability to perform in its tracking and avoidance tasks. This impact is most evident in the behaviour of the agent in the dual agent system. In the case where both agents were avoiding each other the horizontal boundary of the environment gave rise to the counterintuitive effect of both agents ending up catching each other. It is also worth noting that the restriction of the vertical height of the environment could have meant that agents would have had less time to foveate objects and consequent lower performance had the agents not been evolved to perform as adequately possible in the environment's size.

It is also interesting that the agent's evolution to perform the discrimination task led to an agent whose default behaviour was to move continuously in one direction. This represents an example of minimally cognitive behaviour in that instead of the agent being capable of discriminating between a line, a circle and no object at all and then acting appropriately in three different modes, the agent only perceives two classes of object: circle or not circle and reacts in only one of two ways post discrimination.

However, the agent's evolved construction was more complex than absolutely necessary. The fact that the performance of the agent would remain more or less unhindered with the removal of a couple of neurons suggests that the agent was not constructed as simply as it could be for the cognitive behaviour that was being investigated: there was some redundancy in its neuronal architecture. The corollary of the SED perspective on cognition, which states that no single component is responsible for the agent's performance, would be that no component of the agent would be inconsequential. These results therefore appear to be in disagreement with the standard SED perspective. One reason for this could have been the evolutionary process by which the agent's neuronal architecture was derived. It is possible that if

the agent had been evolved so that an even greater fitness value had been achieved, there would be no insignificant neurons in the agent's nervous system.

The neuronal redundancy in the agent's nervous system, where at least two neurons could be removed without significantly impacting the agent's performance, would not have occurred in an agent with bilaterally symmetric brain. Furthermore, it is possible that the bilateral asymmetry of the agent's neural architecture rendered the agent's performance more sensitive to noise on its sensor neurons. It is the author's belief that a bilaterally symmetric agent would have 'spread' the effect of noise on it's sensor neurons more effectively and hence been less likely to erroneously perceive lines as circles.

The agent's performance was relatively robust when its sensor and motor neurons were subject to noise. This was not true of the line avoiding agent when noise was applied to its sensor neurons – the sensitivity here is because the noise applied to the agent's sensor neurons fools it into tracking what it thinks is a circle instead. The line/circle classification of the agent is clearly not robust and the two classes evidently have a very fine line separating them.

# 12. Conclusion

The experiments described within this report have been successful in explaining the sometime counterintuitive and complex behaviour of the agent in the framework of the SED perspective. Furthermore, they make it possible to accept or refute the hypotheses made at the beginning.

The generalisation ability of simple agents to track objects was good for objects dropped within the ranges of offsets used in the agents' evolution and did turn out to be poor outside these ranges. The same was true for the more complex object discrimination agents. The evolutionary approach adopted resulted in agents that were (almost) as simple as they could be for the tasks they were meant to perform and no better. Had wider ranges been included in the agent's evolutionary fitness function, it is the author's belief that agents with different CTRNN parameters would have been evolved but that would have been no less successful in their performance over the increased range.

The neuronal response of the agent evolved to discriminate between classes of objects was very different depending on the class of object being perceived. However, it was not hypothesised just how sensitive the agent's class discrimination would be. The manner within which the agent moved in order to track objects did prove to be different to the way it moved to avoid them but this hypothesis had not taken into account the idea of a default movement - which the latter agent mode proved to be.

The emergent behaviour of dual agent systems was non-linear and literally mathematically complex, though it did not turn out to be as inscrutable as initially envisaged.

The loss of bilateral symmetry in the agent's nervous system appeared to make the agent more sensitive to noise and also produce redundancy in its architecture.

However, this is just speculation and a suggestion for further work would be to test this assumption by comparing bilaterally symmetric and asymmetric agents.

# References

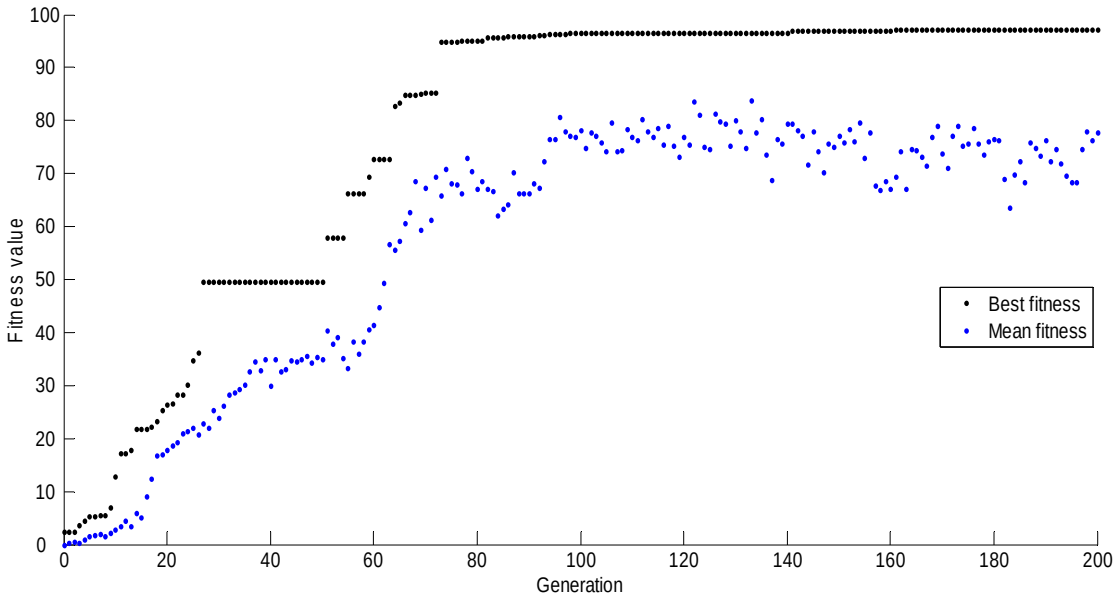
Beer, R. D. (1996) Towards the Evolution of Dynamical Neural Networks for Minimally Cognitive Behavior. In P. Maes, M. Mataric, J. Meyer, J. Pollack and S. Wilson (Eds.), *From animals to animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior* (pp. 421-429). MIT Press.

Beer, R.D. (1997). The dynamics of adaptive behavior: A research program. *Robotics and Autonomous Systems* **20**:257-289.

Slocum, A.C., Downey, D.C. and Beer, R.D. (2000). Further experiments in the evolution of minimally cognitive behavior: From perceiving affordances to selective attention. In J. Meyer, A. Berthoz, D. Floreano, H. Roitblat and S. Wilson (Eds.), *From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior* (pp. 430-439). MIT Press.

# Appendix A

*The fitness value as a function of the number of generations for the evolution of a simple orientation agent.*



# Appendix B

*ConstrainedGaussianMutation.m*

```
function mutationChildren =
ConstrainedGaussianMutation(parents,options,GenomeLength,FitnessFcn,state,thisScore,th
isPopulation,scale,shrink)

% set boundary constraints (this part changed when there were more parameters being
evolved)
LB = [-6    -6    -6    -6    -6    -6    -6    -10   -10   -10];
UB = [ 6     6     6     6     6     6     6     10    10    10];

% set scale of Gaussian
scale = 1;

C = zeros(1,length(LB));

for i=1:length(parents)

    %Choose old parent
    parent = thisPopulation(parents(i),:);

    %Generate random vector
    A = scale .* randn(1,length(parent)) – scale / 2 + parent;

    %'Reflect' values which are over constraints
    D = A + max(LB - A, C) .* 2 + min(UB - A, C) .* 2;

    %Ensure this doesn't make it go over other constraint boundary
    mutationChildren(i,:) = min(max(D,LB),UB);

end
```

*OrientationEvolve.m*

```
function performance = OrientationEvolve(input)

% translate inputs
w(1,:) = input(1:7);
g_sensors = input(8);
bias_sensors = input(9);
bias_motors = input(10);

y_agent = 0;
X_Start = 160:20:240;

%set parameters
h=0.1;
tau = 1;
dia_obj = 26;
dia_agent = 30;
gain_motors = 1;

for a = 1:7
    w(2,a) = w(1,8-a);
end

%Initialise Sensor Neurons
sensor_old = zeros(7,1);
motorNeuron_oldA=0;
motorNeuron_oldB=0;

NumberTrials = 5;

y_obj_velocity = -10;
x_obj_velocity = 0;

for TrialNumber = 1:NumberTrials

%initialise locations/velocities
x_obj = X_Start(TrialNumber);
y_obj = 275;
x_agent = 200;

    while (y_obj - dia_obj/2) > 0

        % Get Sensor values
        I = sensor_intensity(x_obj, y_obj, x_agent, y_agent, dia_obj, dia_agent);

        %Update Sensor Neurons
        for a =1:7
            sensor_new(a) = euler_sensor(sensor_old(a),h,tau,I(a));
        end

        %Update motor Nuerons
        motorNeuron_newA =
euler_motor(motorNeuron_oldA,sensor_new,h,tau,w(1,:),bias_sensors,g_sensors);
        motorNeuron_newB =
euler_motor(motorNeuron_oldB,sensor_new,h,tau,w(2,:),bias_sensors,g_sensors);

        %Update motors
        motorA = sigma(motorNeuron_newA,bias_motors,gain_motors);
        motorB = sigma(motorNeuron_newB,bias_motors,gain_motors);

        %Move Agent
        x_agent_velocity = (motorA - motorB) * 5;
        x_agent = x_agent + h * x_agent_velocity;

        %Move Object
        x_obj = x_obj + h * x_obj_velocity;
        y_obj = y_obj + h * y_obj_velocity;

        sensor_old = sensor_new;
        motorNeuron_oldA = motorNeuron_newA;
        motorNeuron_oldB = motorNeuron_newB;

    end
```

```
    distance(TrialNumber) = abs(x_obj - x_agent);
end

performance = sum(distance) / NumberTrials;

function [sensor_intensity] = sensor_intensity(x_obj, y_obj, x_agent, y_agent,
dia_obj, dia_agent)

rad_obj = dia_obj/2;
rad_adj = dia_agent/2;
Y = y_obj - y_agent - rad_obj;
X1 = x_agent - x_obj - rad_obj * 0.7071;
X2 = x_agent - x_obj + rad_obj * 0.7071;

tanTheta(1) = 0.2679;
tanTheta(2) = 0.1317;
tanTheta(3) = 0.0875;
tanTheta(4) = 0;
tanTheta(5) = -0.0875;
tanTheta(6) = -0.1317;
tanTheta(7) = -0.2679;

cosTheta(1) = 0.9659;
cosTheta(2) = 0.9914;
cosTheta(3) = 0.9962;
cosTheta(4) = 1;
cosTheta(5) = 0.9962;
cosTheta(6) = 0.9914;
cosTheta(7) = 0.9659;

funcSinTanTheta(1) = 0.1986;
funcSinTanTheta(2) = 0.1145;
funcSinTanTheta(3) = 0.0799;
funcSinTanTheta(4) = 0;
funcSinTanTheta(5) = -0.0951;
funcSinTanTheta(6) = -0.1488;
funcSinTanTheta(7) = -0.3373;

for a = 1:7
    %if (tan(theta(a)) > X1/Y) & (tan(theta(a)) < X2/Y)
     if (tanTheta(a) > X1/Y) & (tanTheta(a) < X2/Y)
        %can see object
        %Distance = (Y / cos(theta(a))) + (dia_obj * (1-sin(theta(a))) *
tan(theta(a))) - dia_agent/2;
        Distance = (Y / cosTheta(a)) + (dia_obj * funcSinTanTheta(a)) - rad_adj;
        if Distance <= 0
            %object touching or 'inside' agent
            sensor_intensity(a) = 10;
        else if Distance < 221
                sensor_intensity(a) = min(10, 10/Distance);
            else
                %object too far away
                sensor_intensity(a) =0;
            end
        end
   else
        %can't see object
        sensor_intensity(a) = 0;
    end
end

function [y_new_sensor] = euler_sensor(y_old_sensor, h, tau, I)
y_new_sensor = y_old_sensor + (h/tau) * (I - y_old_sensor);

function [y_new_motor] = euler_motor(y_old, sensor_row, h, tau, w_row, bias,gain)
W = 0;
for a = 1:7
    W = w_row(a) * sigma(sensor_row(a),bias,gain) + W;
end

y_new_motor = y_old + (h/tau) * (-y_old + W );

function [sigma] = sigma(y,bias,gain)
sigma = 1 / (1+exp( (- y - bias)*gain) );
```

*ObjectDiscriminateInterNeuronsEvolve.m*

```
function performance = ObjectDiscriminateInterNeuronsEvolve(input)

% translate inputs
WIn(1,:) = input(1:7);
WIn(2,:) = input(8:14);
WIn(3,:) = input(15:21);
WOut(1,:) = input(22:26);
WOut(2,:) = input(27:31);
g_sensors = input(32);
bias_sensors = input(33);
g_interneurons = input(34:38);
bias_interneurons = input(39:43);
bias_motors(1) = input(44);
bias_motors(2) = input(45);

y_agent = 0;
X_Start = 140:10:240;

%set parameters
h=0.1;
tau = 1;
dia_obj = 30;
dia_agent = 30;
gain_motors = 1;

%Bilateral symmetry
for a = 1:7
    WIn(5,:) = WIn(1,8-a);
    WIn(4,:) = WIn(2,8-a);
end

%Initialise Sensor Neurons
sensor = zeros(7,1);
interNeuron = zeros(5,1);
motorNeuronA=0;
motorNeuronB=0;

NumberTrials = 9;

y_obj_velocity = -5;
x_obj_velocity = 0;

for TrialNumber = 1:9

    obj_type = 0;

    %initialise locations/velocities
    x_obj = X_Start(TrialNumber);
    y_obj = 275;
    x_agent = 200;

    while (y_obj - dia_obj/2) > 0

        % Get Sensor values
        I = sensor_intensity(x_obj, y_obj, x_agent, y_agent, dia_obj,
dia_agent,obj_type);

        %Update Sensor Neurons
        for a =1:7
            sensor(a) = euler_sensor(sensor(a),h,tau,I(a));
        end

        %update inter Neurons
        for i = 1:5
            interNeuron(i) =
euler_IN(interNeuron(i),sensor,h,tau,WIn(i,:),bias_sensors,g_sensors);
        end

        %Update motor Neurons
        motorNeuronA =
euler_motor(motorNeuronA,interNeuron,h,tau,WOut(1,:),bias_interneurons,g_interneurons)
;
```

```
        motorNeuronB =
euler_motor(motorNeuronB,interNeuron,h,tau,WOut(2,:),bias_interneurons,g_interneurons)
;

        %Update motors
        motorA = sigma(motorNeuronA,bias_motors(1),gain_motors);
        motorB = sigma(motorNeuronB,bias_motors(2),gain_motors);

        %Move Agent
        x_agent_velocity = (motorA - motorB) * 5;
        x_agent = x_agent + h * x_agent_velocity;

        %Keep agent within bounds of environment
        x_agent = max(0,x_agent);
        x_agent = min(400,x_agent);

        %Move Object
        %x_obj = x_obj + h * x_obj_velocity;
        y_obj = y_obj + h * y_obj_velocity;

    end

    distance(TrialNumber,1) = abs(x_agent - x_obj);

    %initialise locations/velocities
    x_obj = X_Start(TrialNumber);
    y_obj = 275;
    x_agent = 200;
    sensor = zeros(7,1);
    interNeuron = zeros(5,1);
    motorNeuronA=0;
    motorNeuronB=0;

    obj_type = 1;

    while (y_obj - dia_obj/2) > 0

        % Get Sensor values
        I = sensor_intensity(x_obj, y_obj, x_agent, y_agent, dia_obj,
dia_agent,obj_type);

        %Update Sensor Neurons
        for a =1:7
            sensor(a) = euler_sensor(sensor(a),h,tau,I(a));
        end

        %update inter Neurons
        for i = 1:5
            interNeuron(i) =
euler_IN(interNeuron(i),sensor,h,tau,WIn(i,:),bias_sensors,g_sensors);
        end

        %Update motor Neurons
        motorNeuronA =
euler_motor(motorNeuronA,interNeuron,h,tau,WOut(1,:),bias_interneurons,g_interneurons)
;
        motorNeuronB =
euler_motor(motorNeuronB,interNeuron,h,tau,WOut(2,:),bias_interneurons,g_interneurons)
;

        %Update motors
        motorA = sigma(motorNeuronA,bias_motors(1),gain_motors);
        motorB = sigma(motorNeuronB,bias_motors(2),gain_motors);

        %Move Agent
        x_agent_velocity = (motorA - motorB) * 5;
        x_agent = x_agent + h * x_agent_velocity;

        %Keep agent within bounds of environment
        x_agent = max(0,x_agent);
        x_agent = min(400,x_agent);

        %Move Object
        %x_obj = x_obj + h * x_obj_velocity;
        y_obj = y_obj + h * y_obj_velocity;
    end
```

```
    distance(TrialNumber,2) = min(abs(x_agent - x_obj),50);

end

performance = (sum(distance(:,1)) + 450 - sum(distance(:,2)))/4.5 - 100;


function [sensor_intensity] = sensor_intensity(x_obj, y_obj, x_agent, y_agent,
dia_obj, dia_agent, obj_type)

rad_obj = dia_obj/2;
rad_adj = dia_agent/2;
Y = y_obj - y_agent - rad_obj;

cosTheta(1) = 0.9659;
cosTheta(2) = 0.9914;
cosTheta(3) = 0.9962;
cosTheta(4) = 1;
cosTheta(5) = 0.9962;
cosTheta(6) = 0.9914;
cosTheta(7) = 0.9659;

tanTheta(1) = 0.2679;
tanTheta(2) = 0.1317;
tanTheta(3) = 0.0875;
tanTheta(4) = 0;
tanTheta(5) = -0.0875;
tanTheta(6) = -0.1317;
tanTheta(7) = -0.2679;

if obj_type == 0

    %object is a circle
    X1 = x_agent - x_obj - rad_obj * 0.7071;
    X2 = x_agent - x_obj + rad_obj * 0.7071;

    funcSinTanTheta(1) = 0.1986;
    funcSinTanTheta(2) = 0.1145;
    funcSinTanTheta(3) = 0.0799;
    funcSinTanTheta(4) = 0;
    funcSinTanTheta(5) = -0.0951;
    funcSinTanTheta(6) = -0.1488;
    funcSinTanTheta(7) = -0.3373;

    for a = 1:7
        %if (tan(theta(a)) > X1/Y) & (tan(theta(a)) < X2/Y)
         if (tanTheta(a) > X1/Y) & (tanTheta(a) < X2/Y)
            %can see object
            %Distance = (Y / cos(theta(a))) + (dia_obj * (1-sin(theta(a))) *
tan(theta(a))) - dia_agent/2;
            Distance = (Y / cosTheta(a)) + (dia_obj * funcSinTanTheta(a)) - rad_adj;
            if Distance <= 0
                %object touching or 'inside' agent
                sensor_intensity(a) = 10;
            else if Distance < 221
                    sensor_intensity(a) = min(10, 10/Distance);
                else
                    %object too far away
                    sensor_intensity(a) =0;
                end
            end
       else
            %can't see object
            sensor_intensity(a) = 0;
        end
    end

else

    %object is a line
    X1 = x_obj - x_agent + rad_obj;
    X2 = x_obj - x_agent - rad_obj;

    for a = 1:7
        if (tanTheta(a) > X1/Y) & (tanTheta(a) < X2/Y)
            Distance = (Y / cosTheta(a));
            if Distance <= 0
```

```
                %object touching or 'inside' agent
                sensor_intensity(a) = 10;
            else if Distance < 221
                    sensor_intensity(a) = min(10, 10/Distance);
                else
                    %object too far away
                    sensor_intensity(a) =0;
                end
            end
        else
            %can't see object
            sensor_intensity(a) = 0;
         end
    end

end

function [y_new_sensor] = euler_sensor(y_old_sensor, h, tau, I)
y_new_sensor = y_old_sensor + (h/tau) * (I - y_old_sensor);

function [y_new_motor] = euler_motor(y_old, interNeuron, h, tau, w_row, bias,gain)
W = 0;
for a = 1:5
    W = w_row(a) * sigma(interNeuron(a),bias(a),gain(a)) + W;
end

y_new_motor = y_old + (h/tau) * (-y_old + W );

function [y_new_IN] = euler_IN(y_old, sensor, h, tau, w_row, bias,gain)
W = 0;
for a = 1:7
    W = w_row(a) * sigma(sensor(a),bias,gain) + W;
end

y_new_IN = y_old + (h/tau) * (-y_old + W );

function [sigma] = sigma(y,bias,gain)
sigma = 1 / (1+exp( (- y - bias)*gain) );
```

*DualAgentSystem.m*

```
function DualAgentSystem(input)

% translate inputs
WIn(1,:) = input(1:7);
WIn(2,:) = input(8:14);
WIn(3,:) = input(15:21);
WOut(1,:) = input(22:26);
WOut(2,:) = input(27:31);
g_sensors = input(32);
bias_sensors = input(33);
g_interneurons = input(34:38);
bias_interneurons = input(39:43);
bias_motors(1) = input(44);
bias_motors(2) = input(45);

y_agent = 0;
X_Start = 160:2:240;
L = length(X_Start);

%set parameters
h=0.1;
tau = 1;
dia_obj = 30;
dia_agent = 30;
gain_motors = 1;

%Bilateral symmetry
for a = 1:7
    WIn(5,:) = WIn(1,8-a);
    WIn(4,:) = WIn(2,8-a);
end

%Initialise Sensor Neurons
sensor_one = zeros(7,1);
sensor_two = zeros(7,1);
interNeuron_one = zeros(5,1);
interNeuron_two = zeros(5,1);
motorNeuronA_one=0;
motorNeuronB_one=0;
motorNeuronA_two=0;
motorNeuronB_two=0;

y_obj_velocity = -2.5;

for TrialNumber = 1:L

    %initialise locations/velocities
    x_obj = X_Start(TrialNumber);
    y_obj = 275;
    x_agent = 200;

    elapsedTime = 0 - h;
    g = 0;

    while (y_obj - dia_obj/2) > 0

        g = g + 1;
        elapsedTime = elapsedTime + h;

        % Get Sensor values
        I_one = sensor_intensity(x_obj, y_obj, x_agent, y_agent, dia_obj,
dia_agent,1);
        I_two_temp = sensor_intensity(-x_agent, -y_agent, -x_obj, -y_obj, dia_agent,
dia_obj,1);
        for f = 1:7
            I_two(f) = I_two_temp(8-f);
        end

        %Update Sensor Neurons
        for a =1:7
            sensor_one(a) = euler_sensor(sensor_one(a),h,tau,I_one(a));
            sensor_two(a) = euler_sensor(sensor_two(a),h,tau,I_two(a));
```

```
        end

        %update inter Neurons
        for i = 1:5
            interNeuron_one(i) =
euler_IN(interNeuron_one(i),sensor_one,h,tau,WIn(i,:),bias_sensors,g_sensors);
            interNeuron_two(i) =
euler_IN(interNeuron_two(i),sensor_two,h,tau,WIn(i,:),bias_sensors,g_sensors);
        end

        %Update motor Neurons
        motorNeuronA_one =
euler_motor(motorNeuronA_one,interNeuron_one,h,tau,WOut(1,:),bias_interneurons,g_inter
neurons);
        motorNeuronB_one =
euler_motor(motorNeuronB_one,interNeuron_one,h,tau,WOut(2,:),bias_interneurons,g_inter
neurons);
        motorNeuronA_two =
euler_motor(motorNeuronA_two,interNeuron_two,h,tau,WOut(1,:),bias_interneurons,g_inter
neurons);
        motorNeuronB_two =
euler_motor(motorNeuronB_two,interNeuron_two,h,tau,WOut(2,:),bias_interneurons,g_inter
neurons);

        %Update motors
        motorA_one = sigma(motorNeuronA_one,bias_motors(1),gain_motors);
        motorB_one = sigma(motorNeuronB_one,bias_motors(2),gain_motors);
        motorA_two = sigma(motorNeuronA_two,bias_motors(1),gain_motors);
        motorB_two = sigma(motorNeuronB_two,bias_motors(2),gain_motors);

        %Move Agent
        x_agent_velocity = (motorA_one - motorB_one) * 5;
        x_agent = x_agent + h * x_agent_velocity;

        %Move Object
        x_obj_velocity = (motorA_two - motorB_two) * 5;
        x_obj = x_obj + h * x_obj_velocity;
        y_obj = y_obj + h * y_obj_velocity;

        %Keep agent within bounds of environment
        x_agent = max(0,x_agent);
        x_agent = min(400,x_agent);

        %Keep object within bounds of environment
        x_obj = max(0,x_obj);
        x_obj = min(400,x_obj);

        elapsedTimePlot(g) = elapsedTime;
        separation(TrialNumber,g) = x_obj - x_agent;

    end

end

hold on;
for r = 1:L
    plot(separation(r,:),vertPosObject(r,:),'b');
end

hold off;
```

**N.B.** The functions `sensor_intensity`, `euler_sensor`, `euler_motor`, `euler_IN` and `sigma` are the same as those used in *ObjectDiscriminateInterNeuronsEvolve.m* so are not repeated here.