Evolving dynamical system models of path integration.

Robert J. Vickerstaff
Declaration

I hereby declare that this thesis has not been submitted, either in the same or different form, to this or any other university for a degree.

Signature:
Acknowledgements

Many thanks to all the people who have provided help during the last four years. To my parents. To all the members of the CCNR for friendship and support. To Linc, Ian, Andy and Miguel for help on the technical side. For Kyran and Matt for genetic algorithm ideas. To Simon for mathematical input. To Tom and Paul for help and advice regarding path integration, especially to Tom for suggesting this as the topic of my DPhil. To my supervisor Ezequiel. To Phil, Mick and the BBSRC for funding. To Inman and Neil, my examiners, for the improvements and useful references they suggested. Thanks also to Randall Beer for his workshop at Sussex University, which provided the inspiration and tools for my dynamical systems analysis, and to the workshop organisers Eduardo and Fernando.
Preface

Some of the material in this thesis has been previously published. Vickerstaff (2003) encompasses the work presented in Chapter 4, Vickerstaff and Di Paolo (2005b) presents the work contained in Chapter 5 and elements of the detailed analysis of the experiment 2B model of this chapter are also published as Vickerstaff and Di Paolo (2005a). All three papers should be considered as my own work.
Evolving dynamical system models of path integration.

Robert J. Vickerstaff

Summary

This thesis uses a genetic algorithm to produce the first evolved model of the animal navigation behaviour known as path integration. Comparisons are made to the behaviour of the desert ant *Cataglyphis fortis*. The evolved models produced for this work consist of artificial neural networks, of the types known as Continuous Time Recurrent Neural Networks (CTRNN) and Modified CTRNNs (ModCTRNN), the latter type being introduced in this work. ModCTRNN is found to be better able to evolve solutions under the conditions tested.

The best evolved network is analysed in detail and found to resemble the Mittelstaedt path integration model, and to use a geocentric Cartesian home vector. It is found that accurate homing behaviour occurs even though the network’s memory uses leaky integrators, due to a novel leakage correction mechanism. The network also augments the Mittelstaedt model to generate an efficient searching behaviour at the expected nest location. The imposition of leaky integrators in the standard Mittelstaedt model is found to reproduce the systematic homing errors found in desert ants.

A hand built network is presented which performs path integration using egocentric Cartesian coordinates. A separate (geocentric Cartesian) hand built network is presented which not only performs homing and search, but can also remembers the location of a food item in the arena and returns there later using path integration.

Networks were evolved using a variable length encoding scheme which allowed the automatic pruning of redundant network components during evolution. This enables the resulting networks to be analysed more easily. An extension of this scheme is suggested which retains this feature, but also incorporates special purpose mutation operators designed to allow evolutionary search to explore neutral networks in the fitness landscape.

Submitted for the degree of D. Phil.

University of Sussex

December, 2006
3.2.1 CTRNN ......................................................... 69
3.2.2 ModCTRNN .................................................. 70
3.2.3 CiTRuS ......................................................... 71

3.3 Evolving CTRNN and ModCTRNN with GeNGA .................... 73
3.3.1 CTRNN with GeNGA ....................................... 75
3.3.2 ModCTRNN with GeNGA .................................... 76

3.4 A More Principled Approach to Evolving Variable Topology Networks .................... 77
3.4.1 Custom Mutation Operators .................................. 77
3.4.2 Exploitation of Neutral Networks .......................... 78
3.4.3 Smooth Mutations ............................................ 79
3.4.4 Final Remarks ................................................ 81

4 One Dimensional Path Integration 82
4.1 Overview ......................................................... 82
4.2 Methods .......................................................... 82
4.2.1 The Two Tasks ............................................... 84
4.2.2 The Fitness Function ......................................... 84

4.3 Results .......................................................... 86
4.3.1 The Two-way Task ............................................ 86
4.3.2 The One-way Task ............................................ 91

4.4 Conclusions ....................................................... 97

5 Two Dimensional Path Integration. I 98
5.1 Overview .......................................................... 98
5.2 Methods .......................................................... 99
5.2.1 GeNGA ......................................................... 99
5.2.2 The Path Integration Task .................................. 99
5.2.3 Calculating Fitness ........................................... 100
5.2.4 The Agent ...................................................... 100
5.2.5 Experiment 1A ............................................... 102
5.2.6 Experiment 1B ............................................... 102
5.2.7 Experiment 2A ............................................... 102
5.2.8 Experiment 2B ............................................... 102
5.2.9 Experiment 3A ............................................... 103
5.2.10 Experiment 3B ............................................. 103
5.2.11 Experiment 4 ............................................... 103
5.2.12 Experiment 5 ............................................... 103

5.3 Results .......................................................... 103
5.3.1 Experiment 1A: CTRNN constant-speed PI ................ 103
5.3.2 Experiment 1B: CTRNN variable-speed PI ................ 108
5.3.3 Experiment 2A: ModCTRNN constant-speed PI ........ 108
5.3.4 Experiment 2B: ModCTRNN variable-speed PI .......... 108
5.3.5 Experiment 3A: ModCTRNN constant speed with linear cosine 131
5.3.6 Experiment 3B: ModCTRNN variable speed with linear cosine 131
5.3.7 Experiment 4: ModCTRNN constant speed with head direction cells 131
5.3.8 Experiment 5: ModCTRNN constant speed with positive cosine 132

5.4 Discussion ....................................................... 132
## Contents

6 Two Dimensional Path Integration. II 133  
6.1 Overview 133  
6.2 A CiTRuS Controller for PI Using Rotation Rate Sensors 133  
6.2.1 The Design of the Agent 133  
6.2.2 Behaviour of the CiTRuS Agent 134  
6.3 Conclusions 137  

7 Vector Navigation 139  
7.1 Overview 139  
7.2 Sketch of the Vector Navigation System 140  
7.3 Detailed Definition of the Vector Navigation System 141  
7.4 Behaviour of the Vector Navigation System 143  
7.5 Conclusions 145  

8 Discussion 148  
8.1 Conclusions Relating to Path Integration 148  
8.1.1 The One Dimensional Tasks 148  
8.1.2 PI in Two Dimensions 149  
8.2 Conclusions Relating to Neural Networks 151  
8.2.1 Comparison of CTRNN to ModCTRNN 151  
8.2.2 CiTRuS 152  
8.3 Conclusions Relating to Genetic Algorithms 153  
8.3.1 Automatic Network Pruning 153  
8.3.2 Generality Versus Specialisation 153  
8.4 Conclusion 153  

A Glossary and Abbreviations 163  

B Mathematical Glossary 164  
B.1 Logarithms 164  
B.2 Differentiation 164  
B.3 Integration 165  
B.4 Trigonometry 165  
B.5 Miscellaneous 165  

C Reversibility of the Embedded Two Variable Egocentric Polar System within the Mittelstaedt Model 166  

D Leaky integration model for navigation errors after long straight journeys 168  

E Derivation of leakage corrected PI models for all four coordinate systems 170  
E.1 Leakage Corrected Geocentric Cartesian 170  
E.2 Leakage Corrected Geocentric Polar 171  
E.3 Leakage Corrected Egocentric Cartesian 171  
E.4 Leakage Corrected Egocentric Polar 172  
E.5 Conclusion 173  

F Proof of equivalence of the four leaky PI models 174  
F.1 Geocentric Equations 174  
F.2 Egocentric Equations 175  
F.3 Linking the geo- and egocentric leaky models 175  
F.4 Conclusions 176
G  Equivalence of Rotating Agent to Linearly Damped Pendulum  177
List of Figures

2.1 The four main classes of possible home vector. Left, geocentric rectangular \((x,y)\) and polar \((r,\gamma)\), right, egocentric rectangular \((x',y')\) and polar \((r',\gamma')\). \(N\) is the nest position, \(A\) is the animal’s position, the short arrow indicating its orientation. \(\theta\) is the animal’s orientation relative to the \(x\)-axis. We use the convention of measuring angles as positive anti-clockwise from the \(x\)-axis or from the animal’s body axis. 30

2.2 Outline of the turning dynamics of an animal following the homing system defined by Mittelstaedt and Mittelstaedt (1973), assuming the forward speed \(s\) is zero, expressed using (A) geocentric Cartesian or (B) geocentric polar coordinates. See text for symbol definitions. The arrows under the graphs show the direction that \(\theta\) will move in as the animal rotates, and hence show graphically how the heading has one stable (homeward) and one unstable value for all HV values aside from \((0,0)\). 32

2.3 The PI data and model from Müller and Wehner (1988). Data show the homing direction (phi) of the ant after walking along an L-shaped path with first section of length 10m, second of 5m, separated by the angle shown (alpha). Continuous line, correct homing direction; large diamonds (○), data from ant experiments; crosses (×), Müller and Wehner’s model; pluses (+), the model without the 1.57 scaling factor but with the approximations of sine and cosine (see text); squares (□), the model with the 1.57 scaling factor but with ordinary (non-approximate) sine and cosine functions. The scaling factor accounts for more of the difference between the correct and the ant’s observed homing angle than does the inclusion of the function approximations. 42

2.4 Comparison of Müller and Wehner (1988) with leaky integrator model. Data showing the homing error (epsilon, deviation from homeward heading, degrees) of the ant after walking along an L-shaped path with first section of length 10m, second of 5m, separated by the angle shown (alpha, degrees, clockwise). Positive values of epsilon indicate that the ant has turned too far. Large diamonds (○), data from ant experiments; pluses (+), Müller and Wehner’s model; squares (□), leaky integrator model (see text) fitted to ant data. 42

2.5 The Mittelstaedt model with \(k = 0.1\), initial conditions \(x = 0, y = 1, \theta = 0\). The starting position is marked by an \(S\). 47

2.6 The Mittelstaedt model with \(k = 1\), initial conditions \(x = 0, y = 1, \theta = 0\). The starting position is marked by an \(S\). 48

2.7 The Mittelstaedt model with \(k = 9\), initial conditions \(x = 0, y = 1, \theta = 0\). The starting position is marked by an \(S\). 48

2.8 The Mittelstaedt model with \(k = 30\), initial conditions \(x = 0, y = 1, \theta = 0\). The starting position is marked by an \(S\). 49

2.9 Phase portrait of the two variable dynamical system embedded in the egocentric polar form of the Mittelstaedt PI system. \(r'\) is the nest’s distance from the agent, \(\gamma'\) is the bearing to the nest from the agent. This system has two fixed points, marked by dots, at (\(\sqrt{\frac{1}{k}}, \frac{\pi}{2}\)) and (\(\sqrt{\frac{1}{k}}, -\frac{\pi}{2}\)) which are neutrally stable centres. See text for detailed description. 50
2.10 The leaky HV integration process fitted to data (Müller and Wehner, 1988) taken directly from Figs 11a-c in Hartmann and Wehner (1995). The ants walked along a channel from the nest, turned through an angle \(\phi\), walked along a second channel before being released on a test field. The lengths of the channels were (a) 10m and 5m, (b) 5m and 10m and (c) 10m and 10m respectively. \(\phi\) shows the ants’ homing direction (diamonds). The top line shows the leaky model using a time constant \(\alpha = 18.38\), the lower line shows the correct homing angle. 54

2.11 Graph showing the homing distance \(y\) exhibited by ants which have walked an outward distance \(x\) in a straight line. Top line: \(y = x\), i.e. the correct homing distance. Middle lines: two leaky integrator models fitted to data (diamonds) taken directly from Fig. 2 of Sommer and Wehner (2004). The models are Eqn. 2.12 with \(\alpha = 98.27, \beta = 0.0103\) (taken from their paper) and Eqn. 2.13 with \(\alpha = 193.6\) (fitted using least squared error). The models are indistinguishable. Also shown as the bottom line is Eqn. 2.13 with \(\alpha = 18.38\), the value obtained fitting the leaky PI model to the L shaped journey experiments (see text), clearly a poor fit. 55

4.1 The structure of the best evolved CTRNN network performing the two-way distance measuring task: moving forward until the food sensor is activated then returning to the location it began from and signalling. Fo, food sensor; Sp, speed sensor; Fw, forward motor neuron; Rv, reverse motor neuron; Sg, signal neuron. \(w_{X,Y}\) indicates a weight from component X to component Y. For weight and neuron parameter values see Table 4.1. 87

4.2 The dynamics of the best evolved agent for the two-way distance measuring task. Fw, output of the forward motor neuron (left axis); Rv, output of the reverse motor neuron (left axis); Sg, output of the signal neuron (left axis); A, position of the agent relative to the starting position (right axis). Horizontal lines mark the nest position (0, right axis) and the signalling threshold (0.4859, left axis). The agent is visiting and returning from a food item placed at 400 distances units (the maximum experienced during evolution). The agent signals when the signal neuron output crosses the signal threshold in an upward direction approximately at time 836. The agent crosses the nest location approximately at time 837. 88

4.3 Plot of the error in the agent’s signalling location for the original evolved network and five hand-modified versions of the network, as a function of the distance from the starting location to the food. The correct location to signal is zero. A, the unmodified evolved network; B, reverse neuron modified; C, forward neuron modified; D, forward and reverse neuron modified; E, rescue I; F, rescue II. See text for details of modifications applied. 89

4.4 Plot of the signal neuron state (thin line) and the excess offset value (see text) being inputted to the signal neuron (thick line) over the course of a journey to and back from a food item placed at 400 distance units from the nest. The wind speed was set to zero. Were the network to exactly compensate for the CTRNN decay term, the two values would be identical over the entire journey. 92

4.5 The best evolved CTRNN network performing the one-way distance measuring task: moving forward until the food sensor is activated then continuing forward for an equal distance before signalling. Fo, food sensor; Sp, speed sensor; Fw, forward motor neuron; Rv, reverse motor neuron; Sg, signal neuron. \(w_{X,Y}\) indicates a weight from component X to component Y. For weight and neuron parameter values see Table 4.2. 93
4.6 Internal dynamics of the CTRNN network performing the one-way task. The agent is visiting a food item 400 distance units from the nest, then continuing for a further 400 units. Thin jumpy line, output of forward motor neuron (left axis); thick diagonal line, position of agent (right axis); thick curved line, output of signal neuron (left axis); upper horizontal line, correct signalling location at 800 distance units (right axis); lower horizontal line, signal neuron threshold (left axis).

4.7 The location of the agent when it signalled as a function of the distance from the nest to the food item. Zero (left axis) is the correct place to signal. The wind speed was set to zero for all journeys. Lower thin line, the evolved network; lower thick line, the network with weight $w_{Sg,Fw}$ removed; upper thin line, the network with weight $w_{Sg,Sg}$ removed; upper thick line, the network with both $w_{Sg,Fw}$ and $w_{Sg,Sg}$ removed. In all cases the behaviour of the network was unchanged if the reverse neuron and all its incoming and outgoing weights were removed.

4.8 3D plot of the agent’s speed, the forward motor output and the wind speed during a journey of total length 800 distance units. See text for discussion.

5.1 The four compass response functions used. From the bottom: cosine, positive cosine, piecewise linear approximation of cosine, head direction cell. Plot of sensor output (y-axis) against $\theta_A - a$ (x-axis), where $\theta_A$ is the agent’s current orientation and $a$ is the sensor’s preferred direction.

5.2 Experiment 1A CTRNN network solving the constant speed PI task. $B_{L/R}$, left/right beacon sensor; $C_{L/R}$, left/right compass sensor; $R_{L/R}$, left/right rotation motor neuron; $N_{L/R}$, interneurons. Arrows are directional weighted links; double headed arrows indicate two links running in opposite directions. Grey circles are neurons involved in the oscillations responsible for generating the various modes of behaviour see in this agent during its return journey (see Fig. 5.3).

5.3 Experiment 1A CTRNN agent performing the constant speed path integration task. $B_1$ indicates the single beacon. Beacon and nest are draw as circles radius 0.01.

5.4 Experiment 1A CTRNN network dynamics for oscillating neurons $N_{R/L3}$ and $R_{R/L3}$ during return phase of journey (see Fig. 5.3). Plots are of neuron firing rates, the y axis runs from 0 (bottom) to 1 for each. $B_1$ and nest indicate the time of arrival at the beacon and the nest respectively (the nest is reached at the very end of the trial). $N_{L/R3}$, $R_{L/R3}$ are as in Fig. 5.2.

5.5 Final average homing direction (see text) plotted as a function of the location of the beacon visited before homing commenced. The agent started from the nest $(0,0)$ with a random orientation $\in [-\pi, \pi]$. Beacons locations were sampled with a spatial frequency of 0.1 distance units. Each location was tested with 12 agent runs starting from different initial orientations.

5.6 Final average homing direction (see text) plotted as a function of the location of the beacon visited before homing commenced. The agent started from the nest $(0,0)$ with a random orientation $\in [-\pi, \pi]$. Beacons locations were samples with a spatial frequency of 0.1 distance units. Each location was tested with 26 agent runs starting from different initial orientations. In addition, the agent’s orientation and the states of the two oscillatory interneurons and the two rotation control neurons were randomised upon reaching the beacon.

5.7 Fig. 5.5 with five homing trajectories superimposed. Each trajectory begins at the location of the beacon the agent has just visited and continues for 5 time units. The agent’s HV neurons are not frozen after reaching the beacon, nor are any of its other internal states randomised. The agent began each journey from the nest with a random orientation. No sensory or motor noise is present.
5.8 Experiment 2B ModCTRNN network. $B_{L/R}$, left/right beacon sensor, $C_{L/R}$, left/right compass sensor, $R_{L/R}$, left/right rotation motor neuron, $S$, speed sensor and $F$, forward motor neuron. Arrows are weights, double headed arrows are two weights going in opposite directions between the same end points. Lines ending in small squares are weights which modify other weights. $w_{L1/6}$, are weights. See Table 2 for parameter values.

5.9 Experiment 2B Agent (see Fig. 5.8) performing the PI task. $B_{1,2,3}$ indicate the order the beacons were presented in, beacons and nest are drawn as circles of radius 0.01.

5.10 Experiment 2B Network performing the variable speed path integration task. Plot shows weights (y-axis, see Fig. 5.8) over the whole trial (x-axis, see Fig. 5.9). $B_{1,2,3}$ show the approximate time of arrival at each beacon (the nest is reached at the very end of the trial). From about 2 time units into the trial until 3 time units, a large change in the values of $w_{L3/R3}$ reflects noise applied to the forward motor output as detected by the speed sensor. The second abrupt change in these values reflects the period of enforced captivity of the agent at the last beacon before homing begins.

5.11 A 3D $(t,x,y)$ plot of the unmodified agent returning from a beacon placed at $(0,0.5)$. The simulation used the Runge-Kutta 4th order numerical integration method with a timestep size of 0.0001. There are no sources of noise. The simulation lasted 50 time units.

5.12 A $(x,y)$ plot of the same journey as Fig. 5.11, where the unmodified agent returns from a beacon placed at $(0,0.5)$. There are no sources of noise. The simulation lasted 50 time units.

5.13 The experiment 2B agent’s behaviour if the nest is removed (a) with all sources of noise removed and (b) with 1 percent sensor noise, 10 percent rotation motor noise and 70 percent forward motor noise (as during evolution). A searching behaviour is clearly visible which does not rely on the presence of noise, and arises from the coupled dynamics of the agent’s network and motion.

5.14 Experiment 2B Search density of the agent when the nest has been removed, averaged over the final 35 per cent of 50 independent trials. The agent is returning from a single beacon placed at a distance of 0.75 to the right in all trials. The white dot shows the fictive nest position, the black bar shows the size of the nest (diameter 0.02).

5.15 The behaviour of the agent over 100 time units, returning from a beacon at $(0,0.5)$, with the contralateral weight connecting the rotation control neurons changed from $-3.5$ to $-4.5$.

5.16 The behaviour of the agent over 100 time units, returning from a beacon at $(0,0.5)$, with the contralateral weight connecting the rotation control neurons changed from $-3.5$ to $-4.5$. The length of the scale bar is 0.05.

5.17 The behaviour of the agent over 100 time units, returning from a beacon at $(0,0.5)$, with the contralateral weight connecting the rotation control neurons changed from $-3.5$ to $-5.0$.

5.18 The behaviour of the agent over 100 time units, returning from a beacon at $(0,0.5)$, with the contralateral weight connecting the rotation control neurons changed from $-3.5$ to $-5.0$.

5.19 The behaviour of the agent over 100 time units, returning from a beacon at $(0,0.5)$, the rotation control neuron time constant is 0.005 (where the evolved value was 0.01). The length of the scale bar is 0.05 distance units.

5.20 The behaviour of the agent over 100 time units, returning from a beacon at $(0,0.5)$, the rotation control neuron time constant is 0.02 (where the evolved value was 0.01).
5.21 The behaviour of the agent over 100 time units, returning from a beacon at (0, 0.5), the rotation control neuron time constant is 0.04 (where the evolved value was 0.01).

5.22 The homing direction of the agent (diamonds) after visiting beacons marking out an L shaped journey with leg 1 of length 0.5 and leg two of length 0.25. The angle between the two legs is shown along the x axis. The homing direction (radians, clockwise from down) is shown along the y axis. The solid line shows the geometrically correct homing direction, the squares show the homing direction of \textit{C. fortes} from Müller and Wehner (1988).

5.23 The homing direction of the agent (diamonds) after visiting beacons marking out an L shaped journey with leg 1 of length 0.5 and leg two of length 0.5. The angle between the two legs is shown along the x axis. The homing direction (radians, clockwise from down) is shown along the y axis. The solid line shows the geometrically correct homing direction, the squares show the homing direction of \textit{C. fortes} from Müller and Wehner (1988).

5.24 The homing direction of the agent (diamonds) after visiting beacons marking out an L shaped journey with leg 1 of length 0.5 and leg two of length 0.5. The angle between the two legs is shown along the x axis. The homing direction (radians, clockwise from down) is shown along the y axis. The solid line shows the geometrically correct homing direction, the squares show the homing direction of \textit{C. fortes} from Müller and Wehner (1988).

5.25 Part of the network of the agent evolved in experiment 2B which most directly implements the Mittelstaedt bicomponent PI and homing system. \textit{C}_L and \textit{C}_R are compass sensors, \textit{S} is the speed sensor, \textit{R}_L and \textit{R}_R are rotation control neurons. Here weight pair 2 is labelled as normal for the complete network (Fig. 5.8), weight pair 3 is labelled \textit{kS} and weight pair 4 is labelled \textit{k} for reasons explained in the main text.

5.26 A simplified five variable approximation of the full evolved network. The agent was released from the location (0,0.5) and allowed to home and search for 30 time units. The length of the bar is 0.05 distance units. The search pattern is centred on the nest location. This model used the sigmoidal function \( f(x) = \frac{1}{1 + e^{-x}} \) for the neuron activation functions. The search pattern is very similar to that produced by the full network.

5.27 A second simplified five variable approximation of the full evolved network. The agent was released from the location (0,0.5) and allowed to home and search for 30 time units. The length of the bar is 0.05 distance units. The search pattern is centred on the nest location. This model used the linear function \( f(x) = Ax + B \) with \( A = 0.1712, B = 0.4602 \) for the neuron activation functions. The search pattern is again very similar to that produced by the full network, showing that sigmoidal functions are not required to produce it.

5.28 The linear activation function model, with parameter \( A = 0.26 \). The agent, after an initial transient, begins a repeating figure of eight trajectory in \((x,y)\) space. This plot is of \((t,x,y)\) space. The agent was released from the location (0,0.5), this plot shows the behaviour over the first 5 time units.

5.29 The linear activation function model, with parameter \( A = 0.26 \). This shows the same journey as Fig. 5.28. The centre of the figure of eight loop is at coordinates (0.0494,0.0757). This agent was released from (0,0.5). Five other runs were performed from different starting points, all ended in a congruent figure of eight loop at positions on a circle of radius (0.0902), and each was oriented to the nest as this one is, such that the long axis of the eight is orthogonal to the nest direction. The scale bar is 0.005 units long.
5.30 The linear activation function model, with parameter $A = 0.21$. The agent was released from $(0,0.5)$. The scale bar is 0.02 units long. The pattern is centred on the nest location. This shows the agent’s motion from 45 to 50 time units after release. ................................. 129

5.31 The linear activation function model, with parameter $A = 0.19$. The agent was released from $(0,0.5)$. The scale bar is 0.01 units long. The nest location is very close to the point where the trajectory crosses itself towards the bottom right of the picture. This shows the agent’s motion from 20 to 50 time units after release, and consists of multiple loops around the circuit shown. ................................. 129

5.32 The linear activation function model, with parameter $A = 0.12$. The agent was released from $(0,0.5)$. This shows the agent’s motion from 0 to 30 time units after release. ................................. 130

5.33 The linear activation function model, with parameter $A = 0.09$. The agent was released from $(0,0.5)$. This shows the agent’s motion from 0 to 60 time units after release. ................................. 130

6.1 CiTRuS network performing the PI task using rotation rate sensors and an egocentric Cartesian HV. A leakage-corrected leaky HV is employed. The rotation control neurons act to filter the output of the Mittelstaedt homing system to produce search patterns in the same manner as the evolved PI network with compass sensors. $B_{L/R}$, left/right beacon sensor; $C_{L/R}$, left/right rotation rate sensors; $Sp$, forward speed sensor; $R_{L/R}$, left/right rotation control neuron; $X/Y$, egocentric Cartesian HV; $W$, decaying leakage correction value. The bar across the beacon sensor-to-rotation neuron weights indicates that the weight (drawn as if ipsilateral for convenience) is in fact contralateral. Small black dots are CiTRuS m-nodes (which multiply there inputs together). ................................. 135

6.2 Motion of the idiothetic compass agent as it visits three beacons (not shown) and homes using a leaky egocentric Cartesian home vector. The agent starts from the nest at $(0,0)$, but centres its search at approximately $(-0.14,0.11)$. ................................. 136

6.3 Dynamics of the egocentric Cartesian HV storing neurons during the journey shown in Fig. 6.2. Dotted line, internal state of neuron $X$ in Fig. 6.1; continuous line, internal state of neuron $Y$ in Fig. 6.1. ................................. 136

6.4 Average homing error of the idiothetic- (Fig. 6.1) and allothetic-compass (Fig. 5.8) networks. Homing error is defined as the average distance of the agent from the nest during the last 5 time units of a 15 time unit excursion to between 1 and 3 randomly placed beacons (see Section 5.2.2). Plots are averages taken over 100 trials. Noise magnitude is as defined for the evolution of the allothetic-compass network (see Section 5.2.4). Pluses, noise applied to rotation rate sensors of idiothetic network; Crosses, noise applied to speed sensor of idiothetic network; Asterisks, noise applied to compass sensors of allothetic network; Squares, noise applied to speed sensor of allothetic network. Noise applied to the non-varied sensors was fixed at magnitude 0.01. At and above noise magnitude 0.04 for the idiothetic-compass sensors, the idiothetic-network is no longer able to perform phototaxis to the beacons. ................................. 137
7.1 CiTRuS network topology used to implement the vector navigation system. The complete network is bilaterally symmetrical, and is shown here with each bilateral pair of components collapsed down to a single symbol. The bilateral components are: $B_{L/R}$, left/right beacon sensors; $C_{L/R}$, left/right compass sensors (outputting $\cos \theta$ and $\sin \theta$ respectively); $X/Y$, HV integrators; $S/T$, SV (stored vector) neurons; $R_{L/R}$, left/right rotation control neurons. Non-bilateral components are: $Fd$, food sensor; $Ne$, nest sensor; $Sp$, speed sensor; $Fw$, forward speed control neuron; $W$, decaying leakage correction coefficient; $No$, nest sensor not neuron; $Fl$, nest sensor filter neuron; $Sw$, offset mechanism switch neuron. Small black circles are multiplier (m-) nodes, single headed arrows are $\phi$ weights into m-nodes or $w$ weights into neuron $I$ terms, double headed arrows are $w$ weights into neuron $I$ terms. Short lines drawn immediately behind arrow heads indicate that the weight is contralateral rather than ipsilateral. For example the connection from $B_{L/R}$ to $R_{L/R}$ represents: an ipsilateral $\phi$ weight from $B_L$ to an m-node, followed by a contralateral $w$ weight from the m-node to $R_L$; and an ipsilateral $\phi$ weight from $B_R$ to another m-node, followed by a contralateral $w$ weight from this m-node to $R_R$.

7.2 Frequency distribution of the number of trips from the nest to the food and back completed by the agent during 200 time units, with all sensor and motor noise removed. 1000 trials were performed in total. Between one and three beacons were placed at random bearings from the nest at a distance of between 0.75 and 1.0 distance units.

7.3 Frequency distribution of the number of trips from the nest to the food and back completed by the agent during 200 time units, with sensor and motor noise as used during the evolution of the 2B network: forward motor noise magnitude 0.7, rate 2.0; rotation neuron noise magnitude 0.1, rate 20.0; all sensor noise magnitudes 0.01, rates 20.0. 1000 trials were performed in total. Between one and three beacons were placed at random bearings from the nest at a distance of between 0.75 and 1.0 distance units.

7.4 Example trajectory of the agent with noise present. The agent completes 14 round trips before getting lost. After reaching the single beacon marking the feeder by phototaxis, the agent spends the first approximately 60 time units making shuttle runs between nest and feeder. Brief periods of search like behaviour can be seen at both the nest and feeder locations. Between approximately times 60 and 90 the agent is searching for the nest for an extended period. After finding it around time 90 it moves towards to feeder, but underestimates the distance significantly, and spends the remainder of the trial searching for it approximately two-thirds of the way from the nest to the true feeder location.

7.5 Example trajectory of the agent without noise. The agent completes 72 round trips during the 200 time unit trial, and does not get lost. Only the first 40 time units are shown. After visiting the two beacons, the agent travels towards the nest which it narrowly misses and overshoots. The nest is reached after a brief period of search. The agent then moves towards the feeder which again it narrowly misses and overshoots. After another brief search the feeder is found. The rest of the trial shown consists of orderly movement between nest and feeder. Other brief periods of search occur later on in the trial (not shown).
List of Tables

2.1 Summary of symbols used in leaky integrator equations, arranged according to the coordinate type and reference frame used. GC, geocentric (animal’s location relative to the nest). EC, egocentric (nest location relative to the animal’s location and orientation). .......................................................... 54

4.1 Parameter values of the evolved CTRNN network performing the two-way distance measuring task. Fo, food sensor; Sp, speed sensor; Fw, forward motor neuron; Rv, reverse motor neuron; Sg, signal neuron. \( w_{XY} \) indicates a weight from component X to component Y. \( \theta \) is the signalling threshold of the signal neuron. All neurons have the centre-crossing flag \( f \) set. Hence the values shown for \( v_0 \) and \( b \) are the centre-crossing ones. See Fig. 4.1. ................................................. 87

4.2 Parameter values of the evolved CTRNN network performing the one-way distance measuring task. Fo, food sensor; Sp, speed sensor; Fw, forward motor neuron; Rv, reverse motor neuron; Sg, signal neuron. \( w_{XY} \) indicates a weight from component X to component Y. \( \theta \) is the signalling threshold of the signal neuron. All neurons with the flag \( f \) set to 1 are centre-crossing and for these the values shown for \( v_0 \) and \( b \) are the centre-crossing ones. See Fig. 4.5. .......................................... 94

5.1 Summary of results: Net.(network model used): C (CTRNN), M (ModCTRNN); Spd. (speed during trial): const. (constant), var. (variable); Comp.(compass response function, see Fig. 5.1): cos (cosine); lin. cos (linear approximation of cosine), HD cell (head direction cells), pos. cos (positive cosine); Returns (percentage of successful returns to nest out of 1000 test trials); Size (number of interneurons and weights). Results shown are for the best network obtained from three replicate experiments. ......................................................... 104

5.2 Parameter values of the evolved network from experiment 2B. See Fig. 5.8. ........ 110

6.1 Node parameter values of the CiTRuS PI system using rotation rate sensors. Nodes are: \( R_{L/R} \), left/right rotation rate control neurons; \( Fw \), forward speed control neuron; \( X/Y \), egocentric Cartesian HV neurons; \( W \), decaying leakage correction value. The \( g \) parameter indicates the response rate of the node, and is achieved by setting the g-function parameters as follows: \( a^g = b^g = g, c^g = d^g = 0 \) (the variable response rate feature of CiTRuS is thus not utilised here). ...................... 135

6.2 \( \phi \) weight values for the CiTRuS egocentric PI network. All \( \phi \) weights are unity. The weight name indicates the local network topology leading into and out of the weight. The network is bilaterally symmetrical. So, for example, \( X/Y \cdot C_{R/L} \rightarrow Y/X \) indicates a \( w \) weight into neuron \( Y \) from a multiplier (m-)node outputting the product of \( X \) and \( C_R \). The contralateral \( w \) weight, sharing the same magnitude, inputs to neuron \( X \) from an m-node outputting the product of \( Y \) and \( C_L \). .................... 135

7.1 CiTRuS neuron parameters used for the vector navigation network (weight values are shown in Table 7.2). \( R_{L/R} \), left/right rotation control neurons; \( Fw \), forward speed control neuron; \( X/Y \), HV integrator neurons; \( S/T \), SV neurons; \( Sw \), switch neuron; \( Fl \) nest sensor filter neuron; \( No \), nest sensor not neuron; \( W \), leakage correction decay value neuron. ....................................................... 142
7.2 $w$ weight values of the CiTRuS vector navigation system. All $\phi$ weights are set to 1. $\rightarrow$ and $\Rightarrow$ indicate input to the $I$ term and $I^e$ term of the receiving neuron respectively. Dots (·) indicate values multiplied together at a multiplier node before being weighted and inputted to the receiving neuron. $Fw,R_{L/R},X/Y,S/T,F1,NO,Sw,W$ are neurons as defined in Table 7.1. Sensor names are as follows: $B_{L/R}$, left/right beacon sensor; $C_{L/R}$, left/right compass sensor; $Sp$, speed sensor; $Fd$, food sensor; $Ne$, nest sensor.
Chapter 1

Introduction

This chapter begins with a brief introduction to some of the ideas and previous work behind the main methodology used in this thesis, evolutionary robotics. The introduction to the main thesis topic, the animal navigation behaviour known as path integration (PI), is postponed to Chapter 2. This chapter also provides an overview of the thesis structure and a summary of its main achievements.

1.1 Evolutionary Robotics

The methodology of evolutionary robotics (Beer and Gallagher, 1992; Harvey et al., 2005, 1997) is to employ artificial evolution to create a control system (or artificial “nervous system”) for a real robot or for a simulated agent. Morphologies or other key aspects of the agent may also be evolved. The result is an agent which was not designed directly by the researcher and hence is likely to be less influenced by his or her prejudices. The agent can then be analysed, with the aim of understanding how the evolutionary algorithm has solved the task required of the agent. Hence the final outcome of an evolutionary robotics experiment may be new insights into the nature of the cognitive processes required by the agent’s task. This thesis applies the techniques of evolutionary robotics to PI for the first time, and hence is the first time that a model of PI has been produced without the internal workings having been directly designed by, or at least strongly constrained by, the researcher.

1.1.1 The Study of Cognition

We could define cognition here as “thinking, broadly construed,” with the understanding that this must encompass the internal processes behind many types of animal behaviour as well as that of humans. The study of cognition currently has three main “schools of thought”; symbolism, connectionism and dynamicism (Eliasmith, 1996, in press). The dynamical approach (van Gelder, 1998; Beer, 2000), the newest of the three schools, places a greater emphasis on the role of time and on the coupling of the brain to its body and environment. The metaphor that the brain acts like a computer is rejected. This new emphasis, clearly influenced by and related to the cybernetics

---

1 I owe this admirably concise definition to Thomas Busey
movement, can be applied readily to PI behaviour. Indeed some of the earliest applicable models, such as Mittelstaedt (1962); Mittelstaedt and Mittelstaedt (1973), are in the style of cybernetic flowchart diagrams, and explicitly include the feedback loop created by the animal’s nervous system: sensors provide input to the nervous system which controls motor function which controls orientation and location, which controls sensory input. In fact at least three hierarchical feedback loops can be identified for PI behaviour according to Mittelstaedt (2000) (control of orientation, control of location and control of desired location). Later models have tended to neglect this feedback/coupling aspect somewhat, and concentrate on the internal processing part of the system (see Chapter 2). This neglect can lead to inferior results, for example where the model’s output is not in the correct form to control the animal’s muscles without further, unspecified processing steps.

The explicit inclusion of time in the model allows the dynamics of the animal’s movement to be modelled, hence, rather than merely assuming that the PI system, if correctly functioning, can take the animal home after an excursion, we can model the homing trajectory and study the interaction of homing and systematic search behaviour (see Section 2.1.3). This thesis presents the first PI model to encompass both of these behaviours, where previous models have generally dealt with only one or other, or at least assumed they are separate behavioural routines.

1.1.2 Dynamical Systems and Artificial Neural Networks

Mathematically the type of system constituted by an evolutionary robotic agent in a simulated environment, when expressed as a set of differential or difference equations, is a so-called dynamical system (Strogatz, 1994). Hence the tools of dynamical systems theory can often be used to analyse evolved control systems and agents. The analysis of the most successful evolved PI agent in this work (see Section 5.3.4) proved highly successful.

The type of control systems employed in this thesis, the Continuous Time Recurrent Neural Network (CTRNN) in its original form and also two modified versions (ModCTRNN and CiT-RuS), were chosen since the CTRNN is relatively well studied using dynamical systems theory (Beer, 1995, 2006), and has been successfully used for evolutionary robotics work many times since its introduction to this area by Beer and Gallagher (1992). All three network types are expressed in the form of differential equations which operate in continuous time. The CTRNN was also chosen as a starting point for the present work since it has been used successfully to model insect navigation by Dale and Collett (2001), whose evolved CTRNN controllers reproduced a form of landmark navigation.

1.2 Thesis Overview

This section provides an overview of the structure of the thesis.

Chapter 2 introduces PI, along with *Cataglyphis*, the genus of desert ant which is the primary topic of modelling here. Reasons for choosing these species for the study of PI are given, along with a survey of their behaviour and ecology. Of particular note are the accurate homing ability, systematic searching and systematic homing errors. The sensory cues thought to be used for PI are also reviewed. Next PI is defined in a more abstract mathematical way, and four canonical coordinate systems are introduced within which PI is defined with equations. Next the main existing equational models of PI are reviewed, one of which, the Mittelstaedt model, is particularly
important for the later analysis of an evolved PI model. A more detailed treatment of the homing dynamics of this model is then given as a prelude to the later analysis. The properties of a PI system based on leaky integrators are studied next, with the conclusion that the systematic navigation errors of *C. fortis* ants can be accounted for using this system. Finally three neural PI models are introduced followed by a brief review of PI modelling in rodents.

Chapter 3 introduces the neural network models and genetic algorithm methods used in the following experimental chapters. GeNGA (for General Network Genetic Algorithm) is introduced as a method of evolving variable topology networks made up of any kind of components, provided valid topologies can be defined by simple, local (non-global) rules. This is presented as a sufficient, rather than optimal tool for the following experimental work. The ability to apply automatic, selection based, stochastic pruning of redundant network components is highlighted (I am indebted to Kyran Dale for suggesting this method to me). This feature is used later in Chapter 5, the chapter dealing with evolved models of PI in two dimensions, to simplify the evolved networks prior to analysis. Next the three artificial neural network models CTRNN, ModCTRNN (for Modified CTRNN) and CiTRuS (for Continuous Time Recurrent System) are introduced. These are considered as convenient, general purpose, evolvable dynamical systems, all provably capable of approximating any smooth, continuous time dynamical system given sufficient neurons. They are not considered in terms of their accuracy as neurophysiological models of biological neurons (in this respect only CTRNN is readily defensible). The methods used to implement artificial evolution of CTRNN and ModCTRNN with GeNGA are dealt with. CiTRuS is not used to evolve agents in this thesis, but is used for hand designing. Finally a brief diversion is made to consider what additional features a more optimally designed genetic algorithm might include, with particular emphasis on the idea of creating special purpose mutation operators to exploit neutral networks in the fitness space. These features are not implemented in this thesis.

Chapter 4 presents models of a simplified analogy of PI. Here the simulated agent must only measure the distance it has travelled along a straight line. Upon receiving a sensory cue it must then either return to the starting location by moving backwards an equal distance to that which it has just covered or, in an alternative task, must continue moving forward for the same distance as it has already travelled. These tasks are both solved using CTRNN controllers (ModCTRNN and CiTRuS were not used), and both solutions are analysed to show how the networks use the leaky integrator behaviour of CTRNN neurons to acts as distance measuring devices.

Chapter 5 is the main experimental chapter where simulated agents capable of two dimensional PI are evolved using CTRNN and ModCTRNN. The results of the two types of network are compared, with the conclusion that with CTRNN it is significantly harder to evolve a network capable of efficient PI and homing than with ModCTRNN. The best evolved ModCTRNN network is analysed in detail and found to function in a similar way to the Mittelstaedt PI model, but with two novel features. Firstly, the home vector (HV), the agent’s running estimate of its location, is stored on leaky integrators, meaning that one would expect the agent to make systematic navigation errors similar to those seen in the ant. This is not the case since a leakage correction mechanism (the first novel feature) ensures that the agent’s homing behaviour remains accurate. Secondly, the output of the network corresponding to the homing mechanism of the Mittelstaedt model is passed through a further filtering step, implemented by the rotation control neurons, changing the
behaviour of the agent near the nest into the form of a search pattern.

Chapter 6 shows how a CiTRuS neural network can be used to implement PI using a different kind of HV. Where the previous chapter’s HV was in geocentric Cartesian coordinates, this model uses an egocentric Cartesian system. This stores the location of the nest relative to the agent’s position and orientation, rather than the location of the agent relative to the nest. This model was designed rather than evolved, but was constructed to be a translation of the evolved network into the egocentric Cartesian coordinate system, including the features of leakage correction and search generation. This PI system is found to be less accurate at homing than the evolved model, almost certainly due to its high sensitivity to rotation sensor errors.

Chapter 7 implements a designed vector navigation system, where the agent must remember the location of a food item in order to return there later. This uses a feature of CiTRuS which both CTRNN and ModCTRNN lack, that of “latching”. This model uses a geocentric Cartesian HV, and is once again designed to be an extension of the evolved PI system rather than a completely new design. The features of leakage correction and search pattern generation are again shown to be transferable to this system from the evolved one.

Chapter 8 concludes the thesis with discussion of the conclusions reached relating to PI, and to evolutionary robotics, particularly those relating to evolvable neural networks and genetic algorithms.

1.3 Summary

This section summarises the main achievements of this thesis.

• presents the first evolved PI model
• presents the first PI model combining efficient homing and search
• identifies simple method of producing search from the Mittelstaedt PI model: output filtering
• first published account of leaky integration as possible explanation of ants’ systematic homing errors described in Müller and Wehner (1988)
• first to show errors produced by leaky integration can be equivalent in all four canonical coordinate systems
• introduces two new dynamical agent controllers: ModCTRNN and CiTRuS
• directly compares CTRNN with ModCTRNN on one task
• introduces new, general purpose variable network topology genetic algorithm: GeNGA
• identifies method of navigating accurately with leaky integrators: leakage correction
• shows that a vector navigation system can also function using leakage corrected integrators
Path integration (PI) in nature is a process whereby an animal maintains a running estimate of its location relative to some reference point based on a continuous knowledge of its velocity. This is generally classified as a fundamentally different phenomenon from that of the recognition and use of landmarks, since it relies on the collection of information as an animal moves rather than on the information available solely at the present location. PI characteristically allows an animal to travel over unfamiliar or featureless terrain and yet return to the starting point of its journey by a straight, direct course. Dead reckoning is the term used for the same process in the context of human navigation, particularly at sea. The term path integration was introduced by Mittelstaedt and Mittelstaedt (1980); Mittelstaedt (1983). Many animals are known to be capable of PI including crabs (Layne et al., 2003a,b), ants (Wehner and Srinivasan, 1981, 2003), bees (Dyer et al., 2002), spiders (Moller and G"orner, 1994), geese (von Saint Paul, 1982), gerbils (Mittelstaedt and Mittelstaedt, 1980) and humans (Mittelstaedt and Glasauer, 1991). The animal’s estimate of its location is often referred to as the home vector (HV) (Wehner and Srinivasan, 2003), since in many cases animals use PI to aid their return to a nest or refuge (Layne et al., 2003a; Wehner and Srinivasan, 1981; Hoffmann, 1983). PI may be used to give the animal information about its location in flat or uneven (Wohlgemuth et al., 2001, 2002; Grah et al., 2005) terrain. The velocity information can be derived from a variety of sources, so long as it allows the estimation of speed and direction (Wehner and Srinivasan, 2003).

PI can function in the absence of landmarks or trails (including over water for flying insects), or even in some cases in the absence of any externally generated (allothetic) cues. Flying bees seem to use optic flow to assess the distance travelled (Tautz et al., 2004) whereas walking ants probably use step counting or something similar (Wittlinger et al., 2006). Both types of insect are able to detect the pattern of skylight polarisation and, in conjunction with an internal clock, can use this as a compass cue (Homberg, 2004). Rodents can make use of distant landmarks as compass cues when available, but can also perform PI to some extent in complete darkness, showing that they can use a self generated (idiothetic) compass cue.

The following sections present the rationale for the study of PI behaviour in the desert ant Cataglyphis fortis and its relatives, along with a review of the behaviour and ecological context of the species. The rest of the chapter introduces PI in a more abstract, mathematically precise form,
laying the foundations for later chapters, and reviews existing mathematical and neural PI models.

2.1 The Study of Path Integration in *Cataglyphis* Ants

2.1.1 The Advantages of Studying Path Integration in *Cataglyphis* Ants

Insects, whilst interesting in their own right, are also much less complex than most vertebrates, in terms of the number of neurons they possess, but show a surprising degree of behavioural complexity. They may therefore offer the advantage of presenting the simplest available form of a given behaviour, which can then be compared with research performed on more complex animals if necessary. Navigation behaviour, at which social hymenopteran insects often excel, is a case in point. Their navigational abilities (including those of *Cataglyphis* ants) require many mechanisms, including “sensory processing, decision making, memories and in-built routines” (Collett and Collett, 2002) which are readily observable and manipulable as the animals move around their environments foraging. In the field of insect navigation, aside from the honey bee *Apis mellifera*, ants of the genus *Cataglyphis*, particularly *C. fortis* (Forel) and *C. bicolor* (Fabricius), have been intensively studied (Collett, 1992; Wehner, 1992; Collett, 1996; Wehner et al., 1996; Wehner, 1996; Collett and Collett, 2000, 2002; Wehner and Srinivasan, 2003; Wehner, 2003). *C. fortis* ants live and forage over flat desert salt pans, largely devoid of landmarks. This makes PI the insect’s primary navigational aid and also provides a very simple environment for the purposes of experimentation, modelling and simulation. Unlike most experiments with flying bees, the ant’s foraging journeys can be followed and mapped out in detail (Wehner and Srinivasan, 1981). Channels can be used to force the ants to walk in a particular direction (Müller and Wehner, 1988). A special trolley apparatus can be used to systematically manipulate the compass cues available (Fent, 1986).

2.1.2 The Behaviour and Ecological Context of Desert Ants

Species in the genus *Cataglyphis* are thermophilic scavengers (Cerdá and Retana, 2000; Wehner, 2003). Some species are capable of foraging at body temperatures above 50°C and show behavioural, morphological and physiological adaptations to high temperatures (Gehring and Wehner, 1995). *C. fortis* and *C. bicolor* are both scavengers of dead arthropods, which tend to be small and unpredictably located, precluding the need for pheromone trails or cooperative recruitment of multiple workers to food sources (Wolf and Wehner, 2000). Both species are central place foragers, whereby workers will set out from and return to a central nest. Workers forage individually with little interaction occurring between foragers from the same nest. Long legs and fast running allow short foraging times and, coupled with the use of thermal refuges, allow foraging during the hottest parts of the day, even in their Saharan habitat (Gehring and Wehner, 1995; Wehner et al., 1992). PI allows the ant to return to the nest using the shortest path, and therefore reduces time spent in elevated temperatures (the nest temperature is typically less than 30°C (Gehring and Wehner, 1995)), as well as permitting the rapid collection of food items. PI can also aid the ant’s return to a previously visited feeding site (Collett et al., 1999). Individual ants display a property known as sector fidelity, where they will habitually forage in a given direction from their nest. This may allow an ant to become more familiar with any landmarks present in its sector. Foraging activity has
been modeled in *C. bicolor*, where workers search individually for food, including sector fidelity (Harkness and Maroudas, 1983, 1985). PI ability can be demonstrated by training ants to visit a feeder away from the nest, then capturing them at the feeder as they are about to return home with a food item. Ants transferred to an unfamiliar, distant test area are observed to walk to the location that bears the same position relative to the release site as the nest does relative to the feeder, that is the ants behave as if they are not aware of the displacement. Furthermore the ants not only move to the ‘fictive’ nest location, but also perform a characteristic and efficient searching behaviour, known as systematic search, centred on this location, showing that they have a knowledge of the expected direction and distance of the nest from the release site (Wehner and Srinivasan, 1981). This displacement regime forms the basis of many of the experiments performed on the ants since it is known that they cannot be using landmark or olfactory cues to search for their nest in the test area. A wide variety of experiments have been performed investigating various aspects of their behaviour.

### 2.1.3 Systematic Search Behaviour

The systematic search behaviour has been studied in *C. bicolor*, *C. albicans* (Wehner and Srinivasan, 1981) and *C. fortis* (Müller and Wehner, 1994). The search pattern is not a random walk, but rather remains precisely centred on the location the ant expects to find the nest, and consists of a series of loops which cover all directions evenly (relative to the expected nest position). Rather than being a fixed program which is replayed whenever the ant reaches the expected nest location, search is adaptable to the circumstance. The size of the loops increases the longer the search lasts (Wehner and Srinivasan, 1981; Müller and Wehner, 1994), but the loops still return to the centre. The search patterns also increase in size with the length of the ant’s preceding foraging journey, and hence when it is likely to be less certain of the nest location (Wehner, 1992). The shape of the search pattern can also be biased to one direction under experimental conditions which train the ant to expect the nest to be in this direction. This is the so-called open jaw paradigm, where the ant is displaced by a fixed vector upon reaching the feeder, and allowed to home normally, hence (being unaware of the displacement) the results of PI to the feeder and back home from it do not correspond (Wehner et al., 2002). The ants appear unable to remember a different vector for the outward and return journeys, but after training can adapt the search pattern to be asymmetrical towards the true nest location. The role of the systematic search can be considered as compensation for the inherent inaccuracy of PI based navigation (Müller and Wehner, 1994). Since PI does not rely on (nor does it seem to be reset by, Collett et al., 2003) familiar landmarks, small errors made sensing the velocity (or updating the HV) will tend to accumulate in the HV, making it progressively less accurate the longer PI continues. Upon reaching the estimated nest location, in the absence of a completely reliable way of sensing the nest (see Hoffmann, 1983, for an isopod example), the best strategy is not a spiral path, but one which returns to the location where the nest is most likely to be (Wehner and Srinivasan, 1981; Müller and Wehner, 1994).

Patarnello and Carnevali (1989) introduce a simulated agent controlled by a neural network-style model (constructed from discrete time boolean logical gates) which generates search patterns which appear, similarly to the behaviour of the desert ant, to be closely matched to the properties of the environment. In this case the agent, which moves on a grid of discrete squares, must collect
as many food items as possible during a fixed period. The authors show how the control network which results from their simulated annealing algorithm produces a looping search behaviour which matches the spatial scales present in the distribution of food it has encountered during training. This leads to efficient collection of food.

2.1.4 Systematic Homing Errors

Müller and Wehner (1988) performed an extensive series of experiments where ants were trained to visit a feeder by walking along an L shaped channel. Upon reaching the feeder they were transferred to a test area and the direction they adopted for homing was recorded. It was discovered that the ants tended to turn too far, and hence adopted a heading which would have made them recross their outward path before reaching the nest. The error in heading (only the homeward heading was recorded, not the whole return journey nor the location of the search pattern) was found to be a function of the lengths of the two straight sections of the L shape, and of the angle between the two sections. This lead Müller and Wehner (1988) to propose that the ants were not performing PI in a geometrically correct way, and to propose a mathematically simplified algorithm to account for the systematic homing errors. More recently Sommer and Wehner (2004) tested the homing accuracy of ants which had walked down a long straight channel to reach the feeder. Instead of being released in a distant test area after being captured at the feeder, they were released in a test channel which was always longer than the distance they had walked on the outward journey. The ants tended to begin searching, and would centre their search activity in the test channel, at a point closer to the feeder than the nest. Hence they appeared to underestimate the distance they had travelled, and did so as an increasing function of the journey length. The experimenters proposed a leaky integrator model or a logarithmic function as the best descriptions of this error function.

2.1.5 Other Aspects of Navigation Behaviour

In contrast to many ants, *C. fortis* and *C. bicolor* don’t use pheromone trails for navigation. They can use olfactory cues, in conjunction with the detection of wind direction (Wolf and Wehner, 2000, 2005). An ant will aim at a location downwind of a known food source and, once it reaches the odour plume, begin moving upwind until it reaches its goal. This was shown to be more reliable than simply aiming directly at the food source (ibid.) The other major navigational ability of *Cataglyphis* ants, aside from PI, is visual landmark recognition. This is not used in isolation from PI during natural behaviour, but it is possible to separate the two abilities to some extent during experiments. Landmarks can be removed, the ant can be transported to an unfamiliar or a barren location and/or channels can be used to screen the horizon from the ant (Müller and Wehner, 1988). PI can be disabled to some extent by capturing an ant as it is about to enter the nest (Collett et al., 2001), and hence has already “zeroed” its internal HV (although any movements of the ant from its point of release will still be monitored by its PI system).

There appear to be several ways in which insects use landmarks to aid navigation, which can be summarised under four headings: scene recognition, biased detours, beacon aiming and image matching (following Collett, 1996). Scene recognition involves recognising the visual panorama, and is useful for navigation if accompanied by execution of an associated action, such as head-
ing in a particular compass direction which leads towards home (see Bisch-Knaden and Wehner, 2001, for an example). Biased detours are similar, but involve the recognition of a more discrete landmark, which normally appears on a particular side during a journey. If the object now appears in front of the ant, it will consistently detour to the side which brings the object into its familiar spatial relation as it passes by. In this case the tendency to pass the object always on the same side can act to locally suppress the operation of the PI system, such that the direction of travel indicated by the HV is overruled temporarily by the biased detour initiated by recognition of the object. This behaviour is similar to that which occurs when an unfamiliar object is placed in the path of a homing ant. The ant will detour round the object, but this time selecting a random side, before resuming a course determined by its PI system once the object is passed (Schmidt et al., 1992). The results of *ibid.* show that PI continues to operate during the enforced detour (made during a homeward journey), and that the ant adopts a new homeward course as soon as the object is cleared, rather than returning to the line of its previous course. Beacon aiming involves moving straight towards a familiar object seen in the distance. One or a sequence of such objects could be used in this way during a journey to guide the animal. Finally, image matching is similar to scene recognition, except that the insect’s aim is to bring the image falling on its retina to the point where it exactly matches a remembered scene. This is achieved when the insect is in the same location and orientation as it was when the scene was memorised. This strategy is considered to operate over a smaller range, and to be most useful to locate a small inconspicuous location by reference to nearby conspicuous landmarks.

The combination of PI and landmark recognition makes for a powerful and flexible navigation system, but insects are not considered to take this to its logical conclusion - the construction of cognitive maps whereby the location of known landmarks are memorised by reference to the state of the PI system (Wehner, 2003). The abilities displayed are considered explicable by “the flexible use of vectors, snapshots and landmark-based routes” (Wehner et al., 1996) where “landmarks act primarily as signposts that tell insects what particular action they need to perform, rather than telling them where they are” (Collett and Collett, 2002).

### 2.1.6 Sensory Cues used for Path Integration

PI requires both a compass cue and an odometric cue to be constantly available as the animal moves. The pattern of skylight polarisation is considered to be the most important compass cue used by the ant (Labhart and Meyer, 2002), but the sun’s position and the spectral gradients of the skylight are also used (Wehner and Srinivasan, 2003). The azimuthal component of the sun’s position (but not the elevational component) is used to provide compass information, although this suffers from the disadvantages of being unavailable whenever the sun is not directly visible and becoming less accurate the greater the sun’s elevation is (*ibid.*). Skylight polarisation is sensed by the specialised dorsal rim area (DRA) of the compound eyes, where the arrangement of the microvilli of the ommatidia is such as to maximise their sensitivity to polarised light detection (Wehner, 1994). The insects are able to use a small patch of skylight only 5-10° wide from any part of the sky as a compass cue (Wehner and Srinivasan, 2003). Spectral gradients are also useful, especially for disambiguating the polarisation pattern when the sun is low in the sky and the polarisation (but not the spectral) pattern is symmetrical (*ibid.*). The dorsal area (DA) of the
eye, the remainder of the dorsal part of the eye excluding the DRA, is used to perceive the spectral patterns.

The polarisation detecting microvilli of the DRA are arranged in a special configuration. Each ommatidium has two microvilli sets which are arranged orthogonally (at an angle of 90° to each other). This means that if the orientation of the polarisation plane (the e-vector) is parallel to the first set, it will be orthogonal to the second set. Consequently when light is maximally stimulating one set it minimally stimulates the other. The effect of the two sets of microvilli on polarisation sensitive (POL-) neurons in the nervous system is opposite: one set excites a given neuron while the other inhibits it. The overall effect of this arrangement is to render polarisation detection insensitive to the light intensity (but not to the degree of polarisation) (Labhart, 1999). The response of polarisation sensitive neurons as the orientation of the e-vector is rotated is sinusoidal, with two maxima and two minima over the range 0 to 360°, the separation from a maximum to the next minimum is 90°, as is that from a minimum to the next maximum (Homberg, 2004; Labhart and Meyer, 2002; Labhart, 2000). Hence the function resembles a sine squared function rather than sine.

It is not know exactly how the ant gains odometric information, but it seems that optic flow is only of secondary importance (Ronacher and Wehner, 1995; Ronacher et al., 2000). The strongest theory at present is that they use some kind of idiothetic measure such as step counting or a similar efference copy mechanism. Experimental manipulations of leg length prior to homing provide support for this hypothesis (Wittlinger et al., 2006), since lengthening or shortening the legs for the homeward (but not the preceding outward) journey resulted in corresponding over- and underestimation of the homeward distance. Orientation to gravity must also be sensed since the ants can path integrate successfully over uneven routes (Wohlgemuth et al., 2001, 2002; Grah et al., 2005), but how this is accomplished is not currently known (ibid.).

2.1.7 Modelling PI in Cataglyphis fortis

To produce an abstract model of PI behaviour in C. fortis we must first decide what the key features of the animal’s behaviour and ecology are with respect to PI. The animal forages on flat desert salt pans, where there are few landmarks. The nest is also located within the same habitat. The displacement experiments show that PI is the dominant navigation technique being employed, or at the very least that once removed from its familiar foraging grounds PI alone is sufficient for navigation. We can therefore model the ant’s environment as a flat 2D plane devoid of landmarks, trails and boundaries.

The ant can detect allothetic compass cues, in the form of skylight polarisation and spectral patterns, and the position of the sun. This means that it does not have to integrate its rotational movements into an estimated compass heading idiothetically. Since the ants make short, rapid foragering trips, to minimise thermal stress, the sun’s azimuthal position does not change greatly during the course of a single excursion at the latitude inhabited by the ants studied in Tunisia. The animals also do not naturally forage around midday. However, the ants are capable of compensating for the motion of the sun through the sky. The subject of ephemeris function learning is beyond the current scope of this work, and so the external compass cue given in the simulations presented here will be assumed to be fixed over time, corresponding to a sun which does not move
or to an allothetic cue sensor which already carries out the necessary ephemeris compensation. Theoretical and experimental evidence suggests that the compass heading could be available to the nervous system as sinusoidal functions, either in the form \( \sin \) or \( \sin^2 \). Another plausible form would be four half-wave rectified sine waves, similar to interneurons of the cricket cercal system (Miller et al., 1991). This can be used as a way to implement sinusoidal compass cue sensors without invoking negative neural firing rates.

A differential steer wheeled robot corresponds well to the ant’s “motor output” morphology (Zollikofer, 1994b,a,c), since the ant can rotate its body by moving its opposite side legs in opposing directions, or move forward by moving them in the same direction. Additionally the ant usually walks in the direction of its longitudinal body axis as is necessarily the case for a differential steer robot. *Cataglyphis* ant walking speeds during foraging vary between species and even individual ants but in *C. bicolor* can range from approximately 0.2 to 0.08 m sec\(^{-1}\) (Wehner and Srinivasan, 1981, taken from Fig. 7). In *C. albicans-A* the maximum forward speed is around 0.7 m sec\(^{-1}\) and maximum rate of turn approximately 4000 degrees sec\(^{-1}\). If we model *C. albicans-A* using a differential steer robot such that the maximum straight line and rotational speeds have these respective values, we can calculate the necessary distance between the wheels. Assuming both wheels have the same diameter and the same maximum rotation rate in either direction, an axel length of 2 cm is obtained (independent of wheel size). This is the correct order of magnitude for the separation between an ant’s legs. During searching behaviour the ant must perform many turns, but also wishes to cover the ground as fast as possible. Turning has the cost of lowering the forward speed for both the ant and a differential steer agent. An alternative model is to have the agent’s speed and rotation controlled separately, much like a motor car. This is less like the ant’s method of locomotion, since it has the feature of allowing the agent to turn without altering its speed. This allows a slightly simplified PI task, where forward speed can be fixed at a constant value, allowing HV updating to occur with reference only to the compass sensors. The car steer model also allows a slightly more direct comparison with the Mittelstaedt PI model. For these reasons a car steer model is adopted in this thesis, but it should be noted that the searching behaviour resulting may differ somewhat from a differential steer system.

### 2.2 Modelling Path Integration

#### 2.2.1 Four Coordinate Systems

PI in nature is a process whereby an animal maintains a running estimate of its location based on a continuous knowledge of its velocity. In abstract the values representing the animal’s estimate of its location (or in the absence of errors, its exact location) are referred to as the home vector (HV). In the case of motion on a flat two dimensional (2D) plane, as considered in this thesis, the HV must of course contain two numbers to be able to represent any possible location on the plane. There are four basic ways to perform PI, each using a different coordinate system. It is possible to update (i) Cartesian (rectangular) or (ii) polar coordinates defining the animal’s location relative to a fixed reference point (that is geocentrically) or to update (iii) Cartesian or (iv) polar coordinates defining the location of a fixed reference point relative to the animal’s location and heading (that is egocentrically). The four systems therefore use a HV which specifies the animal’s spatial relationship with the reference point (see Fig. 2.1) using geocentric Cartesian,
Chapter 2. Path Integration

30

Figure 2.1: The four main classes of possible home vector. Left, geocentric rectangular \((x, y)\) and polar \((r, \gamma)\), right, egocentric rectangular \((x', y')\) and polar \((r', \gamma')\). N is the animal’s position, A is the animal’s position, the short arrow indicating its orientation. \(\theta\) is the animal’s orientation relative to the \(x\)-axis. We use the convention of measuring angles as positive anti-clockwise from the \(x\)-axis or from the animal’s body axis.

geocentric polar, egocentric Cartesian or egocentric polar coordinates and the PI model consists of the equations required to update these HVs in response to the animal’s current velocity. The most natural way to express the update of both geocentric HVs is in terms of the animal’s linear speed and compass heading. For the egocentric HVs the natural way is in terms of the animal’s linear speed and rotation rate.

The simplest form to express PI in mathematically is continuous time, using differential equations. This follows from the fact that such a formulation remains precisely correct for an arbitrary time varying velocity. A discrete time formulation necessarily divides the journey up into sections of fixed (or at least predictable) velocity, since otherwise a subordinate PI process must track motion between discrete updates. A discrete formulation is entirely adequate provided the time step size is sufficiently small relative to velocity changes, in which case the model simply approximates a corresponding continuous time system. These considerations apply to PI in abstract.

When applying a model to animal behaviour other considerations may come into play. A discrete time HV update process would come naturally to a human navigator, since one does not wish to be manually updating the HV continuously, whereas in an animal’s nervous system the process may be running continuously without interrupting other functions.

We now define the symbols used to describe the animal’s motion and the four possible HV types throughout this thesis (except when dealing with models by other authors). The units of time and distance are, unless we refer to data from real animal behaviour, only relative to a simulated environment, and hence have no natural interpretation as seconds or metres. Angles and rotation rates will always be in radians (except when dealing with models by other authors). We define \(s\) as the animal’s instantaneous linear speed, \(\theta\) as its compass heading (positive anticlockwise from the \(x\)-axis), \(\dot{\theta}\) the first derivative of \(\theta\) with respect to time (i.e. rotation rate), \((x, y)\) the geocentric Cartesian HV, \((r, \gamma)\) the geocentric polar HV, \((x', y')\) the egocentric Cartesian HV and \((r', \gamma')\) the egocentric polar HV (see Fig. 2.1). \(s\) and \(\theta\) in the general case must refer to the magnitude and direction of the animal’s motion rather than its body axis, but since these coincide for all results presented in this work, the distinction will henceforth be neglected. Any dotted symbol, \(\dot{x}\), is
shorthand for the first derivative with respect to time, $\frac{dx}{dt}$.

The following equations define the geometrically correct way to update the HV in each of the four coordinate systems:

\[
\dot{x} = s \cos \theta \\
\dot{y} = s \sin \theta \\
\dot{r} = s \cos(\theta - \gamma) \\
\dot{\gamma} = \frac{s}{r} \sin(\theta - \gamma) \\
\dot{x}' = \dot{\theta}y' - s \\
\dot{y}' = -\dot{\theta}x' \\
\dot{r}' = -s \cos \gamma' \\
\dot{\gamma}' = \frac{s}{r} \sin \gamma' - \dot{\theta}
\] (2.1-2.4)

These equations define the updating of a HV consisting of two variables. It is not possible to record the animal’s exact location on a 2D plane using fewer than two variables without placing constraints on its movement, such as to a straight line. Redundant systems using greater than two variables could be constructed which also gave geometrically exact results, such as distributing the HV across multiple neurons in a neural network. Any deviation from these equations will in general result in erroneous HV updating, and so we can say in this sense that these are the only geometrically correct formulae for PI, and that any other geometrically correct system is correct by means of equivalence in some way to these four. These HV update equations have been introduced by various authors, but are geometrical truisms rather than inventions or models and can be readily derived from first principles: geocentric Cartesian, Mittelstaedt and Mittelstaedt (1973); geocentric polar, adapted from Hartmann and Wehner (1995); egocentric Cartesian, Merkle et al. (2006); egocentric polar, incorrect version in Gallistel (1990) corrected in Benhamou and Séguinot (1995).

### 2.2.2 Homing

These equations define the process of HV updating, but do not describe any way for the HV to influence the animal’s behaviour. Because we wish to take a neuroethological approach to the study of PI, we require a model that explicitly generates a PI mediated behaviour. The simplest, most natural behaviour to study is homing, where the animal returns to its starting point after an initial excursion using its HV. The simplest, most efficient homing trajectory would be, in the absence of obstacles, a straight line from the current location to the home location. In contrast to the process of HV updating there is more than one possible control equation for a given coordinate system, since the fastest way to return home will depend on the agent’s movement capabilities. A simple method is to assume forward speed is fixed, or not controlled, and that only the rotation rate, $\dot{\theta}$, is determined by the homing system. The PI model given in Mittelstaedt and Mittelstaedt (1973) defines such a system, where $\dot{\theta}$ is governed by the current HV state during homing. The model was originally expressed in geocentric Cartesian form (i.e. Eqn. 2.5), but below it is shown in all four coordinate systems. The following sets of equations now define complete dynamical
systems capable of generating homing behaviour provided we specify the animal’s initial HV and its speed over time (see Section 2.3.5 for an investigation of the dynamics of the Mittelstaedt model).

\[\begin{align*}
\dot{x} &= s \cos \theta \\
\dot{y} &= s \sin \theta \\
\dot{\theta} &= k(x \sin \theta - y \cos \theta) \\
\dot{r} &= s \cos(\theta - \gamma) \\
\dot{\gamma} &= \frac{s}{r} \sin(\theta - \gamma) \\
\dot{\theta} &= kr \sin(\theta - \gamma) \\
\dot{x}' &= \dot{\theta} y' - s \\
\dot{y}' &= -\dot{\theta} x' \\
\dot{\theta}' &= kr' \sin \gamma' \\
\dot{\gamma}' &= \frac{s}{r'} \sin \gamma' - \dot{\theta} \\
\dot{\theta}' &= k r' \sin \gamma'
\end{align*}\]

where \(k\) is a positive constant defining the maximum turning rate. This homing system makes the animal turn towards the nest, turning either clockwise or anticlockwise whichever is the smallest
angle to turn through from its current position and heading. This is the most important feature of the equation, since (if we neglect the animal’s forward motion which constantly changes the nest’s bearing from the animal), it provides a stable equilibrium value that $\theta$ will move towards, leaving the animal facing the nest (see Fig. 2.2). The sinusoidal form of the equation has some support from animal experiments (Mittelstaedt, 1964). An alternative homing system might make the animal turn as fast as possible in the correct direction until it was exactly facing the nest, and then maintain a homeward heading, or use some kind of PID control scheme (van de Vegt, 1990). The elegance of the Mittelstaedt model is that the homing equation only requires the HV and $\cos \theta$ and $\sin \theta$, which must already be available in this scheme to update the HV.

These simple three equation systems illustrate the fact that the form the HV is available in influences the natural form of the homing equation. By describing the HV in a certain form, we assume that the homing system only has the same type of HV available with which to function. All four systems will produce exactly the same homing behaviour, but work differently internally. Working in reverse, if we know an animal has a $\dot{\theta}$ sensor but no direct way to measure $\theta$ (the idiothetic case), we might predict it will use an egocentric system, and vice versa, an animal with a way to measure $\theta$ directly (an allothetic compass) might be predicted on this basis to use a geocentric system.

### 2.2.3 Errors

It is useful at this stage to consider the various types of error that may be present in a PI system, and to attempt to classify them roughly. PI is generally subject to the accumulation of errors as soon as we deviate from the integration of geometrically exact equations acting on precise velocity information. In the absence of external reference points (landmarks), there is no way to correct for HV errors, however they arise, therefore they tend to accumulate over time. Hence the subject of error is a key one. We can begin by classifying errors as being either random, systematic or a combination of both, and further classify on the basis of the source of the error, be it the environment, the sensors, the nervous system (including the HV update process and the generation of motor commands for homing) or the actuators.

Imagining a fictitious animal walking on a featureless 2D plane, an example of a random environmental error would be if unpredictable gusts of wind sometimes blew it off its feet for some distance and if the animal had no sensor capable of detecting these displacements. A systematic environmental error would be if the wind always blew from the same direction. A wind which forcibly displaced the animal at a constant rate during the entire journey could certainly be called systematic, since it lacks any element of stochasticity, but a partially predictable wind, which only displaced the animal intermittently, but always in the same direction is a combination of both. Likewise errors in the measurement of $s$, $\dot{\theta}$ or $\theta$ could be either random but unbiased (with a mean of zero), systematic (such as a consistent misreading of $s$ by a deterministic function) or a combination of both. A time lag in the detection of changes in $s$ or $\theta$ could be classified within this scheme as systematic error. No matter how the HV might be stored and manipulated by the nervous system of our fictitious animal, the process of updating it according to the available sensory information might differ from the geometrically accurate form, including both random and deterministic deviations. Likewise with the generation of motor commands. Finally, the
mechanism of controlling the animal’s limbs using the current HV might not cause it to return to its nest or refuge (as we assume it instinctively tends to do after any excursion), and may introduce random and/or deterministic deviations from a logical homeward trajectory.

### 2.2.4 Differences Between the Four PI Systems

Since the four PI and homing systems defined so far are simply different ways of describing the same process, the behaviour they generate will be exactly the same. This applies to the update of the HV during an arbitrary excursion and homing to the nest. Differences may arise for more complex behaviours. For instance the egocentric HVs, if stored in memory when the agent is at a given location, do not contain the right information to enable the agent to return there later (the agent may later attain the same HV but be in a different location), whereas the geocentric HVs do. Introducing errors into the PI system can also lead to differences. Taking the case of random compass sensor errors, if we subject all four systems to the equivalent errors they will still behave the same, but this entails that the perturbations applied to \( \dot{\theta} \) for the egocentric models be equal to the derivative of the perturbations applied to \( \theta \) for the geocentric models. This can lead a “negative autocorrelation” in the \( \theta \) errors (Benhamou and Séguinot, 1995). To see this imagine a random offset is applied to a previously correct sensor output \( \theta \), and that after a short time the offset is removed. The equivalent errors to be applied to a \( \dot{\theta} \) sensor are two equal and opposite transient errors corresponding to the start and end of the offset to \( \theta \). If instead we applied a random perturbation directly to \( \dot{\theta} \) this autocorrelation is removed, and the accuracy of the egocentric systems is degraded significantly relative to the geocentric ones (ibid.). This corresponds to the fact that the geocentric systems have direct access to the animal’s heading, whereas the egocentric ones only have access to its rotation rate.

In effect the egocentric systems maintain not only a running estimate of the animal’s location, but also of its heading, which is therefore also subject to the accumulation of errors. In this sense the egocentric systems are inherently inferior (ibid.). This does not exclude the possibility that an animal which has direct access to compass cues (such as Cataglyphis ants) uses an egocentric HV internally, since the system could first differentiate the compass signal before using it to update the HV.

The two polar systems each require a HV variable (\( \gamma \) or \( \gamma' \)) with toroidal boundary conditions. When considering how these systems could most naturally be implemented neurobiologically we have several options. Firstly, we could simply assume that the equivalent rectangular system is used in place of the polar one. Secondly, we could present the circular variable as two ordinary variables: \( \cos \gamma \) and \( \sin \gamma \). This is still similar to using the equivalent rectangular form since \( x = r \cos \gamma, y = r \sin \gamma \). Thirdly, we could assume some circular neural structure represented the circular variable directly, as Hartmann and Wehner (1995) use. As can be seen from the equations, both \( \dot{\gamma} \) and \( \dot{\gamma}' \) become infinite when the agent is at its home location (\( r = r' = 0 \)). This corresponds to a discontinuous change in the value of \( \gamma \) and \( \gamma' \) respectively as the agent crosses over its current estimated home location. This situation is not altogether unlikely to arise, since the HV is usually assumed to be zeroed when the animal begins foraging and the home location is also the aiming point of the homing system. Even if the animal never exactly reaches the HV state \( r = r' = 0 \), a small magnitude is sufficient to require a very large value for the derivative of the circular HV.
component in order for it to be correctly updated. Neither rectangular system can generate infinite derivatives in this way, since they do not involve the division operation.

2.3 Existing Mathematical Models of Path Integration

For reviews see Maurer and Séguinot (1995), Gallistel (1990) and Redish (1999). Jander’s 1957 German article (Mittelstaedt, 1962, p. 193; Maurer and Séguinot, 1995) proposed the first mathematical description of a process an animal might perform to accomplish PI, based on the time weighted mean angle of the animal’s heading during a journey, relative to a visual directional indicator (lights were used in Jander’s experiments). The model accounts for the behaviour of Formica rufa L. ants in Jander’s experiments, but the task they were performing was simpler than true path integration. They had only to maintain a certain heading relative to various cues. This model also does not lead to a geometrically correct HV (Mittelstaedt, 1985, p. 300). The HV is in fact not a vector, but a single scalar representing the time weighted average angle that the animal has pursued during its journey (consequently it cannot possibly contain sufficient information to solve PI in two dimensions in the general case):

\[
\alpha_r = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} \alpha(t) dt
\]

where the journey considered occurs between times \(t_1\) and \(t_2\), with the animal’s orientation relative to an external cue at time \(t\) being \(\alpha(t)\) and the final time weighted mean angle being \(\alpha_r\). Of course this does not rule out the possibility of the model accounting adequately for an animal’s behaviour under certain conditions, as indeed it did in Jander’s case, (Maurer and Séguinot, 1995). Müller and Wehner (1988) initially employ a distance (rather than time) weighted mean direction model derived from Jander’s in an attempt to model systematic errors in C. fortis PI mediated homing. Their version of Jander’s model does have a true HV with two values, but they reject the model in favour of a more complicated one which they introduce specifically to match their data (this model is described below). Consequently, it appears Jander’s model provides neither an accurate geometrical description of PI in abstract nor a good description of C. fortis PI behavioural data.

2.3.1 The Mittelstaedt Models

The Mittelstaedt bicomponent models (Mittelstaedt and Mittelstaedt, 1973; Mittelstaedt, 1983, 1985, also Mittelstaedt’s 1978 paper in German cited in Mittelstaedt (1985)) are based on a geometrically correct solution to the general PI problem, and were developed to model the homing behaviour of Agelena labyrinthica (Clerck), the funnel-web spider. Part of the model was originally proposed simply as a way for an insect to hold a steady course relative to an external light source, i.e. photomenotaxis (Mittelstaedt, 1962, p. 191), where some other part of the nervous system would provide the desired heading (or “course order” in Mittelstaedt’s terminology). An extension to this system is also proposed whereby the animal can compensate for the motion of the sun during the day, at any time of year and any latitude (Mittelstaedt, 1962, p. 194), and thereby achieve a geocentrically specified compass heading (provided the system receives input from an annual clock, a diurnal clock and a latitude signal). This system can also orient the animal on a vertical plane relative to gravity, in the same way that honey bees orient themselves during waggle
dances on their vertical honey comb. All of this is based on the decomposition of angular variables into two scalar values equal (or proportional) to the sine and cosine of the original variable, hence the term “bicomponent model”. If the course order is $w$ (radians), the current angle to the light source is $\beta$ (defined as $\beta = - \int u \, dt$ where $u$ is the animal’s rotation rate in radians per unit time) then the animal’s desired rotation rate will be:

$$u = K_a (K_c \cos \beta \sin w + K_s \sin \beta \cos w)$$

where $K_a, K_c, K_s$ are free parameters of the model. The system then acts to bring the animal’s heading to the course order.

Regarding actual PI, Mittelstaedt and Mittelstaedt (1973) define a geometrically correct way to determine the bearing of the animal from its starting location after an arbitrary excursion:

$$\tan \beta_{res} = \frac{\int v_i \sin \beta_i \, dt}{\int v_i \cos \beta_i \, dt}$$

where $\beta_{res}$ is the bearing of the animal from the starting location (measured anticlockwise from the x axis), $v_i$ is the speed of the animal during section $i$ of the journey and $\beta_i$ its heading during section $i$ (measured anticlockwise from a line parallel to the x axis in radians). Integration occurs over time as the animal moves. Note that the values of $v_i$ and $\beta_i$ can jump discontinuously if the journey is specified as a set of discrete sections, or they can be changing continuously, either way the HV remains accurate. If this is rewritten in terms of the differential equations needed to update the HV we obtain:

$$\dot{x} = v(t) \cos \beta(t)$$

$$\dot{y} = v(t) \sin \beta(t)$$

which is clearly the definition of a geocentric Cartesian HV. Benhamou and Séguinot (1995) dismiss this model: “[The Mittelstaedt model] is... mathematically correct, but it is of little interest because it cannot be reformulated in egocentric terms.” (p.465). They assume that PI is likely to be a purely egocentric process and therefore “should be thought of as a mechanism which enables the moving animal to update the egocentric vector specifying the head-referred direction... and the distance... to the memorised home location.” (ibid., p.463). Material relating only to hypotheses about mammalian brain structures is cited including Potegal (1972), and hence cannot justify the rejection of a model developed to model spider navigation. In fact the portion of the Mittelstaedt model they reject (see above) is based purely on geometry, and as such is a mathematical truism. It can also be readily expressed in the three other main coordinate systems, including egocentric polar form (see Eqn. 2.8, although the required compass inputs naturally change from $\cos \beta(t), \sin \beta(t)$ to $\dot{\beta}(t), -\dot{\beta}(t)$). Mittelstaedt (1983) suggests, on the subject of coordinate systems, that when sensory input is initially in polar form, it must at some point be transformed into Cartesian before being used to control muscles:

“...visual information processing, at least in its primary stages, appears to be done in neural layers where visual angle is mapped onto neural position, that is, where space is represented
spatially. Eventually, however, the information thus represented must be transformed into scalar variables fit to control a muscle... We need rigorous theoretical alternatives followed by crucial tests if we are to find out when, where, and how that transformation does occur...” (ibid., p.210).

Contrary to the notion that trigonometric functions must impose a great computational cost on the animal (Müller and Wehner, 1988), the transformation from such polar visual inputs into the sine and cosine components suitable for integration in the bicomponent model can be performed without the nervous system ever having to truly instantiate the trigonometric functions. This is because the network is never required to transform the value \( \beta \) (expressed as a firing rate or other property of neural activity) into the values \( \sin \beta, \cos \beta \). \( \beta \) may be represented rather as an activity profile in a neural map of the animal’s light receptors, which is relatively easy to convert into the required form (Mittelstaedt, 1983; Wittmann and Schwegler, 1995). It seems reasonable to assume from the four expressions of PI in the main classes of coordinate system that the simplest way for a nervous system to perform PI would depend on the form of the sensory input, particularly the compass and rotation rate cues available. It seems reasonable to suggest that where allothetic compass cues are present in the form of spatially mapped activity patterns of this kind, geocentric PI of one form or other is a viable strategy for even a modestly sized nervous system (ibid.). It is also interesting to note that my analysis of the Mittelstaedt model in its simplest form (see Section 2.3.5) was most instructive after a transformation into egocentric polar coordinates, despite the model being originally conceived of in geocentric Cartesian form. Hence there is no need to promote exclusively one coordinate system in the study of PI.

The first of two complete behavioural systems defined in Mittelstaedt and Mittelstaedt (1973) uses the output of the HV integrator to drive the turning rate of the animal, in much the same way as the course order drove the turning rate of the 1962 photomenotaxis system. In this model however photomenotaxis is not used (i.e. the animal does not have access to an allothetic compass cue), rather the animal has a rotation speed sensor whose output is integrated, giving it a completely idiothetic azimuthal (compass) sense. The sine and cosine of the azimuth are integrated over the animal’s path. This is equivalent to integrating over time after first multiplying by speed. These integrals are the HV, which then acts like the sine and cosine of the course order in the 1962 model, thus allowing the PI system to effectively generate its own course order directing the animal towards home. An additional feature adds two scalar offsets to the HV before it is fed into the turning control subsystem, enabling the PI system to steer the animal to any memorised location, the coordinates of which are represented by the offset values. (This is exactly the mechanism employed by the vector navigation system introduced in Chapter 7). The turning control subsystem is otherwise the same as the 1962 model, with \( \sin w \) replaced by \( x \) of the HV and \( \cos w \) replaced by \( y \) (where \( w \) was the course order).

A second model is presented (ibid., p.55) which incorporates both an idiothetic and an allothetic compass sense. This is necessary to model animals which possess both of these sensory modes, such as A. labyrinthica (Moller and Görner, 1994). The angular headings output from these are split into sine and cosine components, weighted, then combined by addition of the two sine components and of the two cosine components before being integrated over the animal’s path. The rest of the model is the same as the first.

Mittelstaedt (1983) and Mittelstaedt (1985) apply a bicomponent model incorporating this dual
compass sense feature to behavioural data from *A. labyrinthica* and the house spider *Tegenaria*. The experiments involve situations where there is a conflict between the headings indicated by the two sensory channels. A possible modification to the model is examined and rejected, namely that the bicomponent outputs from each compass sense be integrated separately, and only additively combined at the point of motor command generation. This would have the desirable feature that performance is rendered independent of the azimuth of the external light cue and/or the origin from which the idiothetic heading is defined (Mittelstaedt, 1985, p. 304). The change is also motivated by results showing that the spider appears able to ignore the past history of (and hence presumably integrals derived from) the output from one of the compass sense channels and home exclusively by the other after an experimental manipulation has rendered the former unreliable (Mittelstaedt, 1983, p.206, experiment D). However, further experiments render this solution unworkable (ibid., experiments E, F, D_H). Separate integration of the two compass channel bicomponents (to give a total of four integrals) is therefore rejected, and a process of normalisation is introduced, able to account for all the experimental results. In this scheme the four bicomponent values from the two senses are weighted, additively combined, then normalised by division by a common factor (the magnitude of the vector consisting of the two weighted combined values), before being integrated. Although this accounts well for all experimental results, it removes the independence from the positions of the external light cue and idiothetic compass origin. Now these two values must be initialised to compatible values before the system functions correctly.

This final version of the model, described concisely in Moller and Görner (1994), Appendix, accounts well for the behaviour of the spider, including more recent results (ibid.), but is more complicated than we require here to model the behaviour of *C. fortis* ants. If we leave out the idiothetic azimuthal sense and the normalisation, the memorised location-offset mechanism and the latitudinal, annual and diurnal compensation mechanisms we are left with the geocentric Cartesian home vector, updated according to an allothetic compass (assumed not to require any form of time-based compensation), and a turning rate equation:

\[
\begin{align*}
\dot{x} &= s \cos \theta \\
\dot{y} &= s \sin \theta \\
\dot{\theta} &= k (x \sin \theta - y \cos \theta)
\end{align*}
\]

where (using the naming conventions employed in the rest of this thesis) \(s\) is the animal’s speed, \(\theta\) its compass heading, \((x,y)\) its HV and \(k\) a (free) turning rate parameter. Here we can clearly see that \(\dot{\theta}\), which is sufficient to make the animal rotate towards its goal, can be calculated without resorting to polar coordinates of any kind. This answers the comments of Gallistel (1990) (p.71):

“The polar model is perhaps more intuitively appealing because to set a homeward course, an animal needs to compute the heading it should adopt and the distance it should run... If the current position is in Cartesian coordinates, a further computation is required to derive the bearing and distance of the nest from the representation of the animal’s current position and heading.”

Gallistel then goes on to define a Cartesian PI model which explicitly converts from the \((x,y)\) coordinates back into a polar form, which is assumed to be the required output of a PI system when homing is undertaken.
2.3.2 The Müller-Wehner Model

Müller and Wehner (1988) present data from navigation experiments with *C. fortis* ants and a PI model to fit this data. The model is a pair of difference equations defining the update of a geocentric polar HV:

\[ l_{n+1} = l_n + 1 - \frac{\delta}{90} \]
\[ \phi_{n+1} = \phi_n + k \left( \frac{180 + \delta(180 - \delta)}{l_n} \right) \]

where \( l_n \) is the distance of the ant from the nest measured in unit lengths (which may be taken as the length of an ant’s stride, where one stride is taken per time step) at time step \( n \), and \( \phi_n \) the direction from the nest to the ant, measured in degrees. \( k = 4.009 \times 10^{-5} \). From later papers (Hartmann and Wehner, 1995; Wittmann and Schwegler, 1995) it appears that the following was intended for the first equation:

\[ l_{n+1} = l_n + 1 - \frac{|\delta|}{90} \]

\( \delta \) is the “angular difference between \( \phi_n \) and the direction of the ant’s \((n+1)\)th step” (Müller and Wehner, 1988, Fig. 2 legend). Whether angles are intended positive clockwise or anticlockwise and from which origin is not specified. Here I take angles as positive clockwise from the direction of the y-axis (“up”), w.r.t. the diagrams in *ibid.*. The model is introduced as a derivation of Jander’s time-weighted mean-direction hypothesis. However, the same model is described in Hartmann and Wehner (1995) as being derived from a geometrically correct PI model which has had the functions sine and cosine replaced with approximate ones. The latter explanation is much more consistent with the model than the former. As mentioned above, the paper first introduces a distance-weighted mean-direction model, where \( l_{n+1} \) is as above, but with \( \phi_{n+1} = \phi_n + \frac{\delta}{l_n} \). This is modified to the final version on the basis that it fits the data much better (which indeed it does). In fact the final version of \( \phi_{n+1} \) now prominently contains a polynomial approximation of the sine function, and is otherwise identical to the geometrically correct geocentric polar update equation, accept that the value of constant \( k \) is 1.57 times larger than expected. To see this we must begin from the exact equation as given by Hartmann and Wehner (1995) (but using the symbol conventions from Müller and Wehner (1988) and assuming a step size of one):

\[ l_{n+1} = l_n + \cos \delta \]
\[ \phi_{n+1} = \phi_n + \frac{\sin \delta_n}{l_n} \]

First we notice that this only applies when \( \phi \) and \( \delta \) are in radians (the natural unit in this case), to convert to degrees we must amend Hartmann and Wehner’s second formula thus:

\[ \phi_{n+1} = \phi_n + \frac{180 \sin \delta_n}{\pi l_n} \]

Next we define the approximations to the sine and cosine functions used by Müller and
valid for $\delta \in [-180, 180]$. The factor $2244737.8$ is arrived at by scaling the polynomial $(180 + \delta)(180 - \delta)\delta$ to have a maximum and minimum of magnitude 1. A very similar factor can be arrived at by fitting the polynomial to the sine function in the interval $[-180, 180]$ following Wittmann and Schwegler (1995). Replacing sine and cosine in the exact equations with these we obtain:

$$l_{n+1} = l_n + 1 - \frac{|\delta|}{90}$$

$$\phi_{n+1} = \phi_n + \frac{180}{2244737.8\pi} \frac{(180 + \delta)(180 - \delta)\delta}{l_n}$$

which is identical to Müller and Wehner’s model except that now $k = \frac{180}{2244737.8\pi} = 2.552 \times 10^{-5}$ instead of $4.009 \times 10^{-5}$. Thus Müller and Wehner’s $k$ is bigger than expected by a factor of $\approx \frac{\pi}{2}$ (and is obtained as the product of $\frac{1}{2244737.8}$, $\frac{180}{\pi}$ and $\frac{\pi}{2}$). The model therefore differs from the accurate update equation in three ways: cosine is replaced by an approximation, sine is replaced by an approximation and the values of $\Delta \phi$ are 1.57 times larger than expected. Consequently it is logical to consider the model (however the authors may originally have conceived of it) to be a slight modification of the geometrically correct equations, contrary to Redish (1999) who considers it (in agreement with Müller and Wehner) to be a system which “[k]eep[s] track of the animal’s mean direction home by weighted averages of distance traveled along each direction” (p.74). Benhamou and Ségui not (1995) rewrite the equations in egocentric polar form and find it “amazing to notice that... [Müller and Wehner] found an approximate angular formula which proves to have a partially correct structure when rewritten in egocentric terms”. This is simply a consequence of the equations being so close to the geometrical ones. It can be seen (Fig. 2.3) that modifying the exact equations by adding only the 1.57 factor (retaining the normal sine and cosine functions) accounts for more of the difference between the correct homing angle and the ants’ behaviour than do the function approximations.

A problem which is encountered with both types of polar PI equations (geo- and egocentric) is the division operation. Here, with the Müller-Wehner model, we must divide by $l_n$ in order to calculate $\phi_{n+1}$. If $l_n$ is exactly zero, we obtain an infinite value. The case of $l_n = 0$ has a special significance for PI, since the HV would be set to this value before an excursion is commenced, and the animal’s aim during homing is to bring the HV back to this state. This is indicative of a more general feature of the polar coordinate systems, namely that the value of the angular coordinate must change very rapidly when the scalar coordinate has a small value. The Cartesian systems do not have this feature: the maximum rate of change in the geocentric case being strictly a function of the maximum speed of the animal, in the egocentric case rapid change can be caused by rotation of the animal when far from the nest. Müller and Wehner (1988) does not state how the first iteration
of the model, when \( l_n = 0 \), is to be performed. My simulations set \( \phi \) initially at the correct value for the first leg of the L shape, and used the rule \( \phi_{n+1} = \phi_n \) when \( l_n = 0 \).

A simplified version of my evolved PI system can be used to produce a Cartesian geocentric model capable of generating errors very similar to the ants. This is basically the Mittelstaedt model with a slowly decaying HV. In fact the method used to produce the errors (modification from the exact equations by the addition of a leakage term) can be applied to all four classes of HV (ego- or geocentric, Cartesian or polar) and reproduces the ants errors in exactly the same fashion (see Section 2.3.6).

Müller and Wehner (1988) do not discuss the possibility that their model may not be the only way of accounting for the ant’s systematic homing errors. The ad hoc methodology used to produce the equations of the model certainly does not rule out this possibility. Section 2.3.6 shows that the four canonical HV update equations can be modified by the addition of a simple decay process and that this renders all four systems capable of reproducing the error graphs produced from the experiments of Müller and Wehner (1988). Specifically, the equations for all four coordinate systems show systematic error properties identical to each other, and which can be fitted to the error data. This means that we can select any one of them as an equally valid description of the error graphs. Here I express the leaky HV update process in geocentric polar form, using their symbol conventions:

\[
\begin{align*}
I_{n+1} &= I_n + \cos \delta - \frac{I_n}{\xi} \\
\phi_{n+1} &= \phi_n + \frac{180.0 \sin \delta}{\pi} \frac{I_n}{l_n}
\end{align*}
\]

where all symbols are as in Müller and Wehner (1988), \( \xi = 1838 \) (fitted using least squared error to the ant data from the 10m:5m L-shape experiment) and one iteration (ant step) is assumed to be 0.01m in length. Here the only alteration from the exact equations is the addition of the \( -\frac{I_n}{\xi} \) term. The fit is approximately as good as that of the Müller-Wehner model. Thus we have found at least one equally good explanation for the systematic error data, and it is clear that more experimental data are required to decide which is the better model. We certainly cannot conclude that the ants perform PI “by employing a computationally simple approximation [the Müller-Wehner model]” *ibid.* (p.5287), and it seems premature to attempt to reverse engineer the control system generating the ant’s systematic search pattern by assuming that PI is performed strictly using the Müller-Wehner model (Müller and Wehner, 1994). The model does account reasonably well for various other experimental results from other species, but there are data (even from *C. fortis*) that it does less well at explaining (Maurer and Séguinot, 1995).

### 2.3.3 The Merkle-Rost-Alt Model

Merkle et al. (2006) utilises an egocentric Cartesian PI system, in fact the geometrically correct form, expressed in continuous time, to model the behaviour of (mainly) *C. fortis* ants. Several systematic error generation mechanisms are investigated whereby the basic PI system is modified, and the resulting deviations from accurate homing behaviour are compared to both the ant’s experimentally observed behaviour and the predictions of Müller and Wehner’s model. The basic
Figure 2.3: The PI data and model from Müller and Wehner (1988). Data show the homing direction (phi) of the ant after walking along an L-shaped path with first section of length 10m, second of 5m, separated by the angle shown (alpha). Continuous line, correct homing direction; large diamonds (□), data from ant experiments; crosses (×), Müller and Wehner’s model; pluses (+), the model without the 1.57 scaling factor but with the approximations of sine and cosine (see text); squares (□), the model with the 1.57 scaling factor but with ordinary (non-approximate) sine and cosine functions. The scaling factor accounts for more of the difference between the correct and the ant’s observed homing angle than does the inclusion of the function approximations.

Figure 2.4: Comparison of Müller and Wehner (1988) with leaky integrator model. Data showing the homing error (epsilon, deviation from homeward heading, degrees) of the ant after walking along an L-shaped path with first section of length 10m, second of 5m, separated by the angle shown (alpha, degrees, clockwise). Positive values of epsilon indicate that the ant has turned too far. Large diamonds (◇), data from ant experiments; pluses (+), Müller and Wehner’s model; squares (□), leaky integrator model (see text) fitted to ant data.
egocentric Cartesian model they employ is:

\[ \dot{X} = -\nu + \omega Y \]

\[ \dot{Y} = -\omega X \]

where \((X, Y)\) is the egocentric Cartesian HV, \(\nu\) is the ant’s forward speed and \(\omega\) is the ant’s rotational speed (anticlockwise, in radians per time unit). My symbols for these values in the rest of the thesis are \(s\) and \(\dot{\theta}\) respectively. As a homing control system they use:

\[ \omega = cY \]

(in fact the homing system also includes stochastic processes, but the underlying control mechanism is as stated above). This is identical to the PI and homing system (Eqn. 2.5) presented in Mittelstaedt and Mittelstaedt (1973) when mapped into egocentric rectangular coordinates (Eqn. 2.8), including the homing control equation (a typographical error is present in the ‘in press’ version of Merkle et al. giving \(\omega = -cY\) [Merkle, personal communication]). Merkle et al. (2006) use what they call a physiological control system, by which they mean that the ant’s rotational speed (and potentially also its forward speed) is perturbed by a standard stochastic Ornstein-Uhlenbeck process, such that the ant’s actual rotational speed is its preferred rotational speed plus a random perturbation. The perturbation (like the PI process itself) changes in continuous time, with a parameter to control the time scale of perturbation and another to control the magnitude. This stochasticity is used to model the directionally persistent random walks observed in the ant’s foraging patterns. Their whole treatment is undertaken in continuous time, which removes the need to specify a step size, or to force the break up of journeys into sections of constant velocity.

The choice of egocentric Cartesian PI is justified by the authors by the simplicity of the equations, which are linear in \(X\) and \(Y\). As the author’s state, once \(\omega\) is included as a third variable, and the PI update equations become part of a control system governing the ant’s rotation, the system is no longer linear:

\[ \dot{X} = -\nu + cY^2 \]

\[ \dot{Y} = -cYX \]

obtained by settings \(\omega = cY\) as the authors suggest. Whether the system is easier to work with as a model being linear in \(X\) and \(Y\) is not immediately apparent, but any neural implementation of the system as stated would clearly have to continuously perform the multiplication of the rotation rate and the two HV components in order to bring the ant back to its nest (a “non-linear” operation). As Maurer and Séguinot (1995) point out, “there is no reason that parsimonious mathematical formulations should correspond to parsimonious neurobiological realizations”. As with Benhamou and Séguinot (1995) they consider that egocentric coordinates “appear to be more adequate” for PI modelling, than geocentric ones, presumably because an animal interacts with the world from an egocentric perspective, but fail to justify this claim. The authors (Merkle et al., 2006) review the
other three PI coordinate systems, but rather than treating them all as equally valid and equivalent descriptions of essentially the same process, they choose one as being the most natural, without making a strong case for the choice. At best this choice should be made as an hypothesis about the way an animal solves the problem, and should be experimentally falsifiable.

For comparison with Merkle et al. (2006), my symmetrical version of the egocentric Cartesian equations are shown here (using the symbol conventions of Merkle et al., 2006), as used in my hand built egocentric Cartesian PI models (see Chapter 6). Since most features of an animal’s morphology are symmetrical, including the nervous system, and most sensory and motor organs are present as bilaterally symmetrical pairs, this seems a natural feature to propose (the Mittelstaedt model already has this feature). The assumption is that there are two organs sensing the rotation rate, the output of one being the negative of the other, and that the neural integrators updating the HV are also bilaterally symmetrical:

\[
\dot{X} = -\frac{\nu}{\sqrt{2}} + \omega Y \\
\dot{Y} = -\frac{\nu}{\sqrt{2}} - \omega X \\
\omega = \sqrt{\frac{1}{2}(Y - X)}
\] (2.10)

The authors state (Merkle et al., 2006) that “[a]ll path integration systems, except the one using geocentric polar coordinates, depend only on the values of \(\nu\) and \(\omega\)...”. This is clearly false as far as the abstract mathematical statements of accurate PI are concerned (geocentric Cartesian is most naturally expressed in terms of the animal’s compass heading rather than \(\omega\)), and from the experiments where *A. labyrinthica* is lifted from the web and replaced, thus causing the animal to abandon use of its idiothetic compass but not preventing accurate homing in some cases, (Mittelstaedt, 1985) it is clear that not all animals require \(\omega\) either. Of course one can express both geocentric systems in terms of \(\omega\) by replacing the compass heading terms by \(\int \omega dt\), and likewise \(\omega\) could be replaced in the geocentric ones by the derivative of the compass heading. Thus there is nothing canonical about \(\omega\) over a direct compass sense. One might say that an egocentric description is the natural language for PI when only \(\omega\) but not the geocentric compass heading is available, and vice versa for a geocentric description.

The paper makes an interesting comparison of the systematic errors generated by several alterations that can be made to the accurate PI system, and compares them to the model given in Müller and Wehner (1988). The error mechanisms they investigate are: (LU) a linear underestimation of \(\omega\), (NLMs) a non-linear underestimation of \(\omega\), (NLU) a second type of non-linear underestimation of \(\omega\), (PD) a processing delay in the value of \(\omega\) and (LI) leaky integration of the HV values. The LI model is exactly equivalent to that presented in geocentric Cartesian form in Vickerstaff and Di Paolo (2005b). As is shown in Appendix F, even though the leaky forms (Section F.4) of the canonical PI equations (Eqns 2.1, 2.2, 2.3 and 2.4) do not lead to accurate navigation, all four leaky models are never-the-less exactly equivalent and therefore show the same systematic errors, despite being expressed in different coordinate systems. The findings presented (Merkle et al., 2006) are in agreement with Vickerstaff and Di Paolo (2005b) in that the time constant required to explain the systematic errors shown in Müller and Wehner (1988) is incompatible with that required for those shown in Sommer and Wehner (2004). It should be noted that the ant’s PI system and searching patterns are known to be adaptable to their foraging experiences regardless of
Chapter 2. Path Integration

the age of the ant, as shown by ‘open-jaw’ experiments (Collett et al., 1999; Cheng and Wehner, 2002; Wehner et al., 2002), so we cannot immediately rule out the possibility that the apparent differences are due to adaptations of this kind as suggested in Vickerstaff and Di Paolo (2005b), due to the training of the ants for the various experiments. Obviously, there could also easily be more than one mechanism involved in the generation of systematic homing errors (Merkle et al., 2006).

2.3.4 The Benhamou-Sauvé-Bovet Model

Benhamou et al. (1990) construct an egocentric polar PI model using a system of discrete steps. The main contribution of their paper is to study the effect of sensory errors on the accuracy of the PI process. They find that errors in an idiothetic compass (with no allothetic compass information available) lead to much greater inaccuracies in the HV than when errors are introduced to an allothetic compass, or to the animal’s estimate of its speed. They refer to PI as an egocentric coding process and distinguish it from an exocentric coding process, by which they mean landmark based navigation. This is slightly ambiguous terminology since PI can be achieved without resorting to egocentric coordinates. Their HV update equations are as follows:

\[ \hat{D}_{i+1} = \sqrt{\hat{D}_i^2 + \hat{P}_{i+1}^2 - 2\hat{D}_i \hat{P}_{i+1} \cos(\hat{\omega}_i - \hat{\alpha}_i)} \]

\[ \hat{\omega}_{i+1} = \arctan \left[ \frac{\sin(\hat{\omega}_i - \hat{\alpha}_i)}{\cos(\hat{\omega}_i - \hat{\alpha}_i)} \right] + k\pi \]

where \( D_i \) is the distance and \( \omega_i \) the angle (anticlockwise in radians) of the nest relative to the animal’s position and heading at time step \( i \), \( P_i \) indicates the distance walked in a straight line during step \( i \) and \( \alpha_i \) indicates the angle turned (anticlockwise in radians) at the end of step \( i \), \( k \) is 0 when \( \cos(\hat{\omega}_i - \hat{\alpha}_i) \) is positive, otherwise \( k \) is 1. The hats, \( \hat{D} \), \( \hat{\omega} \) and so on, indicate that the values are estimates subject to errors. In the case where the compass sense was allothetic, a compass sensor was assumed to give the heading \( \hat{\theta} \), subject to errors, and the value \( \hat{\alpha} \) was calculated as the change in \( \hat{\theta} \) between successive steps.

The discrete time formulation used forces the journey to be broken up into straight sections and leads to more complicated HV update equations in comparison to the continuous time version (using my symbols):

\[ \dot{r}' = -s \cos \gamma' \]

\[ \dot{\gamma}' = \frac{s}{r'} \sin \gamma' - \hat{\theta} \]

and also introduces an extra factor into the behaviour of the system, namely the step size. Benhamou et al. (1990) have to experiment in order to determine how to make the error magnitude of the sensory input scale in such a way as to make the model’s behaviour independent of the step size used. This is described as a purely heuristic process. No such accommodation would be required if a continuous time treatment were used, such as that of Merkle et al. (2006). Given
the twisting and highly convoluted nature of many natural *C. fornis* foraging paths, a treatment allowing continuous changes in velocity seems more suitable for this reason alone.

The errors applied to the sensory values were defined as follows:

\[ \hat{\alpha} = \alpha + e_\alpha \]

with \( e_\alpha \sim N(0, \epsilon_\alpha) \)

\[ \hat{\theta} = \theta + e_\theta \]

with \( e_\theta \sim N(0, \epsilon_\theta) \)

\[ \hat{P} = P + e_P \]

with \( e_P \sim N(0, \epsilon_P) \). The following functions were found to make the error properties “practically independent of path discretisation”:

\[ \epsilon_\alpha = C_\alpha \alpha \]

\[ \epsilon_\theta = C_\theta \]

\[ \epsilon_P = C_P \sqrt{P} \]

where the \( C \) values are constants.

### 2.3.5 Behaviour of the Mittelstaedt Bicomponent Model

The following section presents an analysis of the dynamics of motion of an animal homing according to the basic version of the Mittelstaedt PI system. These results will form a basis for the analysis of the evolved PI agent presented in Chapter 5.

The model can path integrate correctly for arbitrary, time varying forward speeds of the animal. Mittelstaedt (2000) suggests using \( s = -(x\cos \theta + y\sin \theta) \) to control the speed, however, we will not analyse such a system here. Instead we will assume speed is constant. This assumption is useful later on since the evolved PI agent moves at an approximately constant speed. Since all scales have arbitrary dimensions, we will further define speed as unity (1.0), but investigate the effect of varying the maximum turning rate parameter \( k \) of the turning rate equation. If we assume the animal is running at full speed in order to return home as quickly as possible, it becomes natural to consider optimising the turning rate to effect efficient homing. The model then becomes:

\[ \dot{x} = \cos \theta \]
\[ \dot{y} = \sin \theta \]
\[ \dot{\theta} = k(x\sin \theta - y\cos \theta) \]

Numerically integrating this system for various initial conditions and \( k \) values shows the animal’s motion usually forms a spirograph like pattern (see Figs 2.5, 2.6, 2.7, 2.8). The only exception
to these patterns appears to be if the initial value of $\theta$ points the animal directly towards or away from the nest, in which case the trajectory is a straight line continuing forever in the same direction, passing directly through the nest if the animal is initially pointing towards it. These straight line trajectories are stable when pointing towards the nest and unstable when pointing away from it. It appears that, aside from these cases, all trajectories consist of an endless sequence of loops, all of the same shape and size during a single trajectory but varying with different $k$ values and initial conditions, and that the animal therefore does not converge onto the nest location over a series of these loops. A special case of the "looping" behaviour appears when the agent circles the nest instead of alternately approaching and receding from it (Fig. 2.6).

Analysis is simplest when converted from geocentric rectangular to egocentric polar form:

$$\dot{r}' = -\cos \gamma'$$

$$\dot{\gamma}' = \frac{1}{r'} \sin \gamma' - \dot{\theta}$$

$$\dot{\theta} = kr' \sin \gamma'$$

since this contains the following embedded two variable system (obtained by substituting for $\dot{\theta}$ in the $\dot{\gamma}'$ equation):

$$\dot{r}' = -\cos \gamma'$$

$$\dot{\gamma}' = \left(\frac{1}{r'} - kr'\right) \sin \gamma'$$

Plots of this system show closed trajectories for the cases where the animal’s geocentric motion is spirograph like, and corresponding straight trajectories for the cases where the animal passes through or directly away from the geocentric origin. $\dot{r}' = 0$ when $\gamma' = \pm \frac{\pi}{2}$ and $\dot{\gamma}' = 0$ when $\gamma' = \pm \pi, 0$ or $r' = \sqrt{\frac{1}{k}}$ giving $(r', \gamma')$ fixed points at $(\sqrt{\frac{1}{k}}, \pm \frac{\pi}{2})$ and $(\pm \sqrt{\frac{1}{k}}, -\frac{\pi}{2})$ (see Fig. 2.9). At these points $\dot{\theta}$ will be $\sqrt{k}$ and $-\sqrt{k}$ respectively. These two fixed points in $(r', \gamma')$ space correspond to
Figure 2.6: The Mittelstaedt model with $k = 1$, initial conditions $x = 0, y = 1, \theta = 0$. The starting position is marked by an S.

Figure 2.7: The Mittelstaedt model with $k = 9$, initial conditions $x = 0, y = 1, \theta = 0$. The starting position is marked by an S.
Figure 2.8: The Mittelstaedt model with $k = 30$, initial conditions $x = 0, y = 1, \theta = 0$. The starting position is marked by an S.

anticlockwise and clockwise circles about the origin in $(x, y)$ space. The Jacobian is:

$$
\begin{pmatrix}
\frac{\delta \dot{r}}{\delta \gamma} & \frac{\delta \dot{r}}{\delta y} \\
\frac{\delta \dot{r}}{\delta x} & \frac{\delta \dot{r}}{\delta \gamma}
\end{pmatrix} = 
\begin{pmatrix}
0 & \sin \gamma \\
-(\frac{1}{k} + k) \sin \gamma & (\frac{1}{k} - kr') \cos \gamma
\end{pmatrix}
$$

which, evaluated at the two fixed points $(\sqrt{\frac{1}{k}}, \frac{\pi}{2})$ and $(\sqrt{\frac{1}{k}}, -\frac{\pi}{2})$, gives:

$$
\begin{pmatrix}
0 & 1 \\
-2k & 0
\end{pmatrix}
$$

and:

$$
\begin{pmatrix}
0 & -1 \\
2k & 0
\end{pmatrix}
$$

respectively. For both matrices the trace $\tau = 0$ and determinant $\Delta = 2k$. Since $k > 0$, $\Delta$ is positive and the linearised version of the dynamical system at the fixed points is therefore classified as a neutrally stable centre. Centres are one of the borderline cases where no firm conclusions can be reached about the behaviour of the full non-linear system at the fixed points, since it may not conform to that of the linearised version. However, as is shown in Appendix C, the non-linear $(r', \gamma')$ system is a reversible system, which leads to the conclusion that the fixed points are indeed centres and that all trajectories sufficiently close to them are closed, as suggested by the numerical integration plots. Converted back into $(x, y)$ space, this means that all loops of the spirograph patterns are exactly the same shape (are congruent). This in turn means that the whole $(x, y, \theta)$ trajectory will be either periodic or quasiperiodic, depending on whether the net change in $\theta$ over the course of one loop is a rational or irrational fraction of $2\pi$ respectively.

Trajectories on the line $\gamma' = (\pm)\pi$ remain there forever with $r$ increasing towards infinity.
Figure 2.9: Phase portrait of the two variable dynamical system embedded in the egocentric polar form of the Mittelstaedt PI system. $r'$ is the nest’s distance from the agent, $\gamma'$ is the bearing to the nest from the agent. This system has two fixed points, marked by dots, at $(\sqrt{1/k}, \frac{\pi}{2})$ and $(\sqrt{1/k}, -\frac{\pi}{2})$ which are neutrally stable centres. See text for detailed description.

Trajectories beginning on the line $\gamma' = 0$ remain there as $r$ decreases to 0 whereupon the value of $\gamma'$ jumps discontinuously to $\pi$. All other trajectories, apart from the two fixed points, appear to be closed loops circling about one of the fixed points.

With respect to a model of PI, the spirograph-like patterns at first glance might be interpreted as search patterns, where the animal compensates for uncertainty in its knowledge of the goal location by searching in the vicinity of its current best estimate. On closer examination this is untenable since the spirograph patterns always encompass a region that extends from the goal at least as far as the location of the animal when it started homing. We would ideally like a model capable of displaying an initial period of efficient homing, where the animal moves to its estimate of the goal location, possibly followed by a search behaviour whose extent would be controllable by a model parameter allowing the system to be tuned to differing degrees of uncertainty. The Mittelstaedt model, as defined for our purposes here, is incapable of such a separation, and it is therefore reasonable to treat the model as a description of homing but not of search. In this respect the spirograph patterns are completely spurious, since we want the agent to return only to the goal location, and we should set the k parameter sufficiently high that the motion generated is approximately a straight line from the initial location to the goal. The motion after this can be ignored since the animal can be considered to have already reached home. Naturally, were we to apply Mittelstaedt’s speed control equation, $s = -(xcos\theta + ysin\theta)$, we would not expect to observe the same spirograph-like behaviour.

The behaviour of the animal with speed control implemented was not studied here in detail since it would be of little help understanding the simulated PI agents in later chapters which move
with constant speed. For completeness, however, it is worth considering briefly how speed control would influence the properties of the model. First consider that \(-(x \cos \theta + y \sin \theta)\) expressed using egocentric polar coordinates is simply \(r' \cos \gamma'\). This tells us all we need to know about the system for our purposes here: that the animal only ever moves closer to home and that the animal moves more slowly the closer to home it gets. The first feature follows from the shape of the cosine function: speed is positive when facing roughly towards home (\(\gamma' \in [-\pi/2, \pi/2]\)), negative when facing roughly away from home (\(|\gamma'| > \pi/2\)) and zero when facing exactly orthogonal to home (\(\gamma' = \pm \pi/2\)). This makes spirograph patterns or anything resembling search patterns of any kind impossible, and we expect the animal to converge slowly on the home location with ever decreasing speed from all initial conditions (including the case where it initially faces exactly away from home).

Only one feature of the spirograph-like patterns appears suitable for a search behaviour, namely the possibility of quasiperiodicity. Such a trajectory, in \((x, y, \theta)\) space, would never begin to repeat itself and would therefore not begin re-searching locations it has already covered. Ideally the model would guarantee that its behaviour was non-repetitive from any starting position, which is not the case here (the \((r', \gamma')\) fixed points are counterexamples since they correspond to circles in \((x, y)\) space).

### 2.3.6 Path Integration with Leaky Integrators

There is some experimental evidence to suggest that leaky integration may be a feature of PI under some conditions (Mittelstaedt and Glasauer, 1991; Sommer and Wehner, 2004; Vickerstaff and Di Paolo, 2005b; Merkle et al., 2006), and also of related neural integration systems in nature (Major et al., 2004a,b)

As explained in Section 5.3.4, dealing with the analysis of an evolved PI system, it is possible for a system to have leaky (decaying) HV values, and yet still perform accurate homing, at least for journeys which do not exceed a certain duration relative to the decay time constant. This requires that the system implement a “leakage correction” system (see Appendix E). If the HV is leaky but the system does not implement leakage correction, navigational errors arise. As shown in Appendix F, despite the fact that leaky integration of the HV update equations renders the system no longer geometrically accurate as PI, it is possible to define leaky versions of the HV update equations in all four coordinate systems such that the errors they display are the same, hence they remain four alternative ways of describing the same process. The four leaky HV update systems are as follows (see Eqn. F.6, Table 2.1):
\[ \dot{u} = -\frac{u}{\alpha} + s \cos \theta \]
\[ \dot{v} = -\frac{v}{\alpha} + s \sin \theta \]
\[ \dot{u}' = -\frac{u'}{\alpha} + \dot{\theta}' - s \]
\[ \dot{v}' = -\frac{v'}{\alpha} - \dot{u}' \]
\[ \dot{z} = -\frac{z}{\alpha} + s \cos (\theta - \zeta) \]
\[ \dot{\zeta} = \frac{1}{z} s \sin (\theta - \zeta) \]
\[ \dot{z}' = -\frac{z'}{\alpha} - s \cos \zeta' \]
\[ \dot{\zeta}' = \frac{1}{z} s \sin \zeta' - \dot{\theta} \]

where \((u,v)\) is the leaky geocentric Cartesian HV, the incorrectly calculated estimate of \((x,y)\), and similarly for the three remaining coordinate systems, \((z,\zeta)\) is the leaky form of \((r,\gamma)\), \((u',v')\) the leaky form of \((x',y')\) and \((z',\zeta')\) the leaky form of \((r',\gamma')\). \(\alpha\) is the time constant of decay. 

As has already been shown in Section 2.3.2, the leaky form of the otherwise geometrically correct HV update equations can reproduce the Müller-Wehner *C. fortis* error graphs. That section dealt only with the angular error in the HV obtained as the ant left the end of the second leg of the L shape and was about to began homing. We must, however, also assume that the decay of the HV values continues during the homeward leg of the journey, since PI itself appears to continue as normal during homing, as suggested by the ant’s behaviour after enforced detours (Schmidt et al., 1992). Rather than deal only with the ant’s initial heading upon beginning to home, we would like to define the homing error in relation to the final location that the ant arrives at after homing is complete, since we can then deal with the two dimensional spatial offset of this location in relation to the true nest position. Fortunately this redefinition of homing error does not change the angular error, but does influence the distance we expect the animal to walk.

We will define a new simplified homing behaviour which neglects the turning dynamics of the Mittelstaedt homing system, referred to as the Simplified Analytic Model in Vickerstaff and Di Paolo (2005b). We assume the ant, upon beginning to home, maintains \(\zeta' = 0\) (meaning that the ant walks straight towards the perceived nest location) and walks forward, stopping when \(z' = 0\). Examining \(\dot{\zeta}'\) above in the egocentric polar leaky equations, we observe that if \(\zeta' = 0\) the ant has no need to turn, making \(\dot{\theta} = 0\), giving \(\dot{\zeta}' = 0\) throughout the homeward journey. In fact \(\dot{\zeta}' = 0\) whenever the ant walks directly towards or away from its estimated nest without turning. This means that the HV decay does not cause the ant’s return path to curve in any way, and hence we can assume that the ant simply walks in a straight line until its leaky HV is zeroed, whereupon it
will move no further. Due to the decay of the $z'$ value during homing, it will not walk as far as we would expect from a simple examination of the magnitude of $z'$ when homing began.

**L-shaped routes**

The extensive series of experiments conducted by Müller and Wehner (1988) testing the homing behaviour of *C. fortis* after L-shaped outward excursions showed that the error in the heading of the ant’s homeward run (i.e. its angular deviation from the correct homeward path) varied systematically with features of the outward journey, specifically the length of the first straight section, the angle turned moving from the first to the second section and the length of the second straight section. The data are reproduced in fig. 11a - c from Hartmann and Wehner (1995) and were used to fit the $\alpha$ time constant of the leaky PI equations to the behaviour of *C. fortis* (see Fig. 2.10). $\alpha$ was fitted to the first of the three experiments, and the remaining experiments were used to check the model’s generalisation. The fit is visually as good as that achieved by the Müller-Wehner model.

**Long straight routes**

Sommer and Wehner (2004) showed that, on straight journeys, *C. fortis* tends to underestimate the length of the return leg as an increasing function of the outward journey length. They fitted several models to this error function and concluded that a leaky integrator function and a logarithmic function ($y = \alpha \log \left( \frac{1 + \beta x}{\beta} \right)$) were the two best descriptions of the data. The leaky integrator equation they fitted to the data was (see Fig. 2.11):

$$y = \alpha (1 - e^{-\beta x})$$  \hspace{1cm} (2.12)

where $x$ is the length of the outward journey to the feeder, $y$ is the centre of the ant’s search pattern at the end of the return journey and $\alpha$ and $\beta$ are free parameters. This is similar to the model presented in Mittelstaedt and Glasauer (1991). Appendix D shows how the leaky PI equations predict the following function, assuming that the integrator time constant is the same for both legs of the journey:

$$y = \alpha \log \left( 2 - e^{-\frac{x}{\alpha}} \right)$$  \hspace{1cm} (2.13)

An alternative form of PI navigation, called discontinuous PI, is suggested by Collett and Collett (2000), whereby, upon reaching the point where homing must begin, the HV is stored in memory and then zeroed. In order to navigate home the animal would now reverse the sense of its compass response functions and proceed by PI to the point where its current HV equalled the stored HV. This is in contrast to continuous PI where, as the Mittelstaedt model assumes, the HV is not zeroed before homing, but is simply updated during homing as before, while the animal navigates to the location where the HV is zero. If the HV decay rate is assumed to be constant, the discontinuous model predicts the relation $y = x$ for the error function regardless of the value of $\alpha$. In this case the shape of the Sommer-Wehner data clearly favours the continuous model over the discontinuous model under the leaky integrator model presented here. Fig. 2.11 shows Eqns 2.12 and 2.13 fitted to the data. The three models are visually indistinguishable. The $\alpha$ time constants arrived at for the L-shaped (18.38) and straight journeys (193.6) are clearly very different (see Fig. 2.11). Consequently, if we wish to use the same model to explain both data sets, we must propose that the effective integrator time constant can vary between experiments according to unknown factors.
Table 2.1: Summary of symbols used in leaky integrator equations, arranged according to the coordinate type and reference frame used. GC, geocentric (animal’s location relative to the nest). EC, egocentric (nest location relative to the animal’s location and orientation).

<table>
<thead>
<tr>
<th></th>
<th>GC True Location</th>
<th>GC Leaky HV</th>
<th>EC True Location</th>
<th>EC Leaky HV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cartesian</td>
<td>((x, y))</td>
<td>((u, v))</td>
<td>((x', y'))</td>
<td>((u', v'))</td>
</tr>
<tr>
<td>Polar</td>
<td>((r, \gamma))</td>
<td>((z, \zeta))</td>
<td>((r', \gamma'))</td>
<td>((z', \zeta'))</td>
</tr>
</tbody>
</table>

Figure 2.10: The leaky HV integration process fitted to data (Müller and Wehner, 1988) taken directly from Figs 11a-c in Hartmann and Wehner (1995). The ants walked along a channel from the nest, turned through an angle \(A\), walked along a second channel before being released on a test field. The lengths of the channels were (a) 10m and 5m, (b) 5m and 10m and (c) 10m and 10m respectively. \(\phi\) shows the ants’ homing direction (diamonds). The top line shows the leaky model using a time constant \(\alpha = 18.38\), the lower line shows the correct homing angle.
Figure 2.11: Graph showing the homing distance $y$ exhibited by ants which have walked an outward distance $x$ in a straight line. Top line: $y = x$, i.e. the correct homing distance. Middle lines: two leaky integrator models fitted to data (diamonds) taken directly from Fig. 2 of Sommer and Wehner (2004). The models are Eqn. 2.12 with $\alpha = 98.27, \beta = 0.0103$ (taken from their paper) and Eqn. 2.13 with $\alpha = 193.6$ (fitted using least squared error). The models are indistinguishable. Also shown as the bottom line is Eqn. 2.13 with $\alpha = 18.38$, the value obtained fitting the leaky PI model to the L shaped journey experiments (see text), clearly a poor fit.

2.4 Existing Neural Models of Ant Path Integration

2.4.1 The Hartmann-Wehner Model

This model (Hartmann and Wehner, 1995) uses neural chains to store and manipulate the HV. A neural chain is a linear or circular chain of neurons whose pattern of firing represents a linear or circular variable respectively, in this case primarily the distance and angular components of a geocentric polar HV. Each chain neuron has two supporting neurons to allow the activity pattern to be stabilised or modified. The distance chain (called the $r$-chain) works like a thermometer. Each chain neuron can be either fully activated or fully inactivated. Activity spreads from one end of the chain to the other as the value increases, and recedes back again as it decreases. Only the position of the boundary of the active region encodes information. The angular chain (called the $\nu$-chain) works similarly, except that the chain is circular and contains a group of adjacent active neurons. The number of active neurons is fixed, but the activity can shift clockwise or anticlockwise around the circle to represent changes to the stored value. The model has five more circular chains. One, the $\lambda$-chain, represents the ant’s current compass heading. The final four chains, called $C^+, C^-, S^+, S^-$ are needed to calculate the value of $\lambda - \nu$ and apply approximations of the trigonometric functions cosine and sine to them. Chains $S^+$ and $S^-$ provide a homing signal. For further details see Hartmann and Wehner (1995).

The model reproduces the systematic navigation errors displayed by C. fortis (Müller and Wehner, 1988) by nature of the approximate PI mechanism it instantiates. Nine free parameters of the model, controlling the relative phases and widths of the activity regions on chains $\nu$ and $\lambda$, and thus the approximation of cos and sin functions, are used to fit the model to the data. It is also possible to make the model produce more accurate PI behaviour by choosing different values for these parameters (Wittmann and Schwegler, 1995).

Since the model separates the polar HV into distance and angular components a potential
discontinuity in $v$ (the angular HV component) arises at the HV zero point: if the animal were to pass over its estimate of the home location the angular component of the HV would have to jump by $180^\circ$. This might cause problems if the network were employed to model the ant’s systematic search behaviour, since PI must continue during searching, which involves regularly returning to the HV zero point (Wehner and Srinivasan, 1981).

Chapman (1998) reports implementing and testing the model on a mobile robot, using approximately 900 simulated neurons. Flaws in the r-chain and homing mechanisms are reported. After modifying the model to correct these flaws, the robot was able to navigate successfully.

2.4.2 The Wittmann-Schwegler Model

This model (Wittmann and Schwegler, 1995) uses a sinusoidal array to represent its geocentric polar HV as a phasor. The distance and angular HV components (called $(r, \phi)$) specify the amplitude and phase respectively of a sinusoidal wave, which is represented spatially along a circular neuronal array. Using $N$ array neurons the activity of neuron $i$ is $k_0[r \cos(\phi + 2\pi i/N)] + b_0$ where $k_0$ and $b_0$ are positive constants. The animal’s compass heading is assumed to be available as a symmetrical single peaked activity pattern on another circular array of neurons. Connections from the compass array to the HV array, modulated by the animal’s current speed, convert the compass information into the same phasor representation as the HV. Simple element wise addition is then enough to update the HV correctly. Recurrent connections within the HV array act as a memory and stabilise the shape of the sine wave. Homing is achieved by comparison of the current compass and HV arrays. For full details see Wittmann and Schwegler (1995).

The ease of performing vector addition with a sinusoidal array is stated to justify its usage in PI. Touretzky et al. (1993) implement a sinusoidal array using a spiking neuron model. The array is a redundant encoding where each unit of the array is made up of 100 spiking neurons, where a high precision can be attained even if each neuron has relatively few distinguishable firing rates and is subject to noise. However Touretzky et al. state that the sinusoidal array has no advantage over a Cartesian encoding unless its ability to perform vector rotation is used, which the Wittmann-Schwegler model does not employ.

Like the Hartmann-Wehner model Wittmann and Schwegler fit their model to the data from *C. fortis* (Müller and Wehner, 1988). The fit is visually as good as the former model’s. Since the network performs exact, error free PI, an extra parameter and mechanism were introduced specifically to produce errors. Whilst fitting to data using a single parameter is arguably stronger evidence for a model than fitting with nine, the model does not generate navigation errors as a result of its fundamental PI mechanism as the Hartmann-Wehner model does.

2.4.3 The Kim-Hallam Model

This model (Kim and Hallam, 2000), aimed at least partly at genus *Cataglyphis* ants, uses a third method to store the HV, similar to the way a traverse board was used in dead reckoning navigation at sea. A ring of neurons each accumulates the distance travelled by the animal in its preferred direction, using input from a light compass, the circle being divided up evenly between the neurons. These neurons act to store the HV, but a set of weights then acts as a function which converts the activations into a geocentric Cartesian representation. A second ring of neurons calculates the
homing direction and distance from this representation.

Parts of the model are only outlined verbally, preventing a more detailed treatment here. In particular the mechanism updating the HV storing neurons from the light compass is not specified, nor is the exact method used to derive the homing direction and distance. The method of driving the agent (a simulated khepera robot) is defined in terms of pseudocode statements rather than neural or motor function equations.

One interesting feature of the model’s behaviour is the dependence of homing accuracy on the number of neurons used in the HV ring and hence on the granularity of the HV. A stereotyped looping behaviour is observed upon reaching the estimated home location which leads the authors to speculate that a combined homing and search generation system would be possible. This is realised in the evolved PI system presented here in Section 5.3.4.

Unfortunately, one of the authors’ motivations for designing the system appears to be erroneous. They conclude that the Mittelstaedt bicomponent model of PI requires the animal’s nervous system to instantiate the cosine, sine, arctan and square root functions. This assertion is based on the assumption that input to the network can be treated as the animal’s compass heading and forward speed, and that the outputs required for homing are the distance and heading to the nest. This leads them to attempt to avoid the use of these four functions to meet the perceived requirements for PI. The definition of the Mittelstaedt model (Mittelstaedt and Mittelstaedt, 1973) and a consideration of the embodied nature of an animal’s “control system” reveals that arctan and square root are not required, since the output of a PI based homing system need only be a turning tendency and optionally a speed control signal (or, alternatively, motor control signals for actuators on the left and right side of the body). Likewise (see Section 2.3.1) the Mittelstaedt model does not necessarily impose the calculation of sine and cosine functions on the nervous system of the animal.

The optical properties of a simple light sensor can be tuned to give a cosine shaped response to a point source of light, or to any symmetrical light distribution in the environment (Mittelstaedt and Eggert, 1989). Non-cosine shaped sensor response functions can also be converted into cosine shaped functions by a simple weighting process (Wittmann and Schwegler, 1995; Mittelstaedt, 1983).

2.5 Path Integration in Rodents

Navigation by PI is studied in rodents, including rats, golden hamsters and gerbils (Mittelstaedt and Mittelstaedt, 1980). In rats it is possible to record electrophysiologically from multiple neurons while an animal is moving around. This has allowed modelling to proceed towards a realistic neural model of PI. Rat recordings led to the discovery of several sets of neurons whose activity is correlated with navigation behaviour, the three most important being head direction, place and grid cells.

Head direction cells (Taube, 1997) have a pattern of firing which reflects the geocentric compass heading of the animal’s head. Place cells, found in the hippocampus and other connected brain areas, have firing which reflects the animal’s spatial location. A place cell will only fire when the animal is in a certain part or parts of an environment. Grid cells, found in the medial entorhinal cortex, also reflect the rat’s location but in a more striking way - their firing occurs in a set of locations evenly spaced across the environment in the form of a hexagonal close packed
grid. There are also cells whose firing reflects combinations of head direction and place (Cacucci et al., 2004) and head direction and grid (Sargolini et al., 2006) locations.

It is interesting to note that the directional tuning properties of both of these ‘hybrid’ cell types is broader than that of the head direction cells. Head direction cells fire in a narrow band of head direction either side of their preferred direction, whereas the hybrid cell types fire in a more sinusoidal shaped fashion. Compare plots D and F in figure 3 of Sargolini et al. (2006). D shows grid-by-direction cells, F shows head direction cells. Also compare plots B and C in figure 4 of Cacucci et al. (2004). These likewise show place-by-direction and head direction cells respectively. This is of interest since sine or cosine shaped functions of head or body direction are predicted as part of the HV update mechanism of several PI models (see Section 2.2).

The recent discovery of the detailed properties of the grid cell-type (Hafting et al., 2005) suggests that the HV information of the rodent PI system may be held by these cells. Although it is not yet clear how PI may be implemented in detail, the spatial map constituted by the grid cell-type is likely to be central (McNaughton et al., 2006).

Here only a brief summary of neural models of rat PI will be given. This body of work is felt to be of less immediate relevance to PI in desert ants than the models introduced above for several reasons.

2.5.1 Differences Between Rodent and Insect PI

One major difference is that the extensive neurophysiological data has allowed models to be both constrained by and make predictions about detailed neural properties. Naturally this data is of no direct relevance to ant PI, since the ants do not share the brain structures involved and have far smaller brains. Place cells are known to be influenced by the visual environment of the rat (McNaughton et al., 1996), and it seems likely that rodents are able to form more sophisticated links between familiar landmarks and their PI system than are insect navigators. In particular it seems they may be able to reset PI upon recognition of a familiar scene (Etienne et al., 2004). There is no convincing evidence that insects can do this (Collett et al., 2003). It is also likely from the behaviour of place cells that rats can maintain more than one frame of reference at any one time (McNaughton et al., 1996; Zinyuk et al., 2000), analogous to the ability to plot locations on two or more separate charts, where the animal is familiar with multiple experimental arenas but does not necessarily know the spatial relation between them. Again, this ability is thought to be absent in the ant, where it appears impossible for them to memorise separate outward and homeward vectors which are not related by a simple sign reversal of the vector components (Wehner et al., 2002). The rat’s head direction cells represent its geocentric heading, which can still be updated correctly in the dark, implying that purely idiothetic input is sufficient, whereas Cataglyphis ants do not naturally forage without access to a skylight compass cue, and act as if lost when the dorsal rim areas of their eyes are painted out.

2.5.2 Neural Models of Rodent PI

Rodent PI modelling has centred on the idea of using an abstract two dimensional sheet or ‘chart’ of place cell neurons to store the HV, with a group of active cells, the activity packet, representing the animal’s current location. In fact this feature is taken directly from electrophysiological
recording from rats (McNaughton et al., 1996). The chart is abstract since place cells maximally active at adjacent spatial locations are not necessarily found to be adjacent in the brain. Indeed, the same place cells can take part in multiple alternative charts. This approach is called map-based path integration (MPI). For example Samsonovich and McNaughton (1997) propose such a model where the activity packet shape is an attractor for the chart. From a random initial position a packet will form at a random location on the chart. The packet attractor is said to be continuous, in that the packet shape is restored after any minor perturbation, but its location is stable anywhere on the chart. While Mittelstaedt (2000) proposes that the mechanism to update the position of the packet may occur extrinsically to the chart and is based on his bicomponent model, Samsonovich and McNaughton (1997) and Conklin and Eliasmith (2005) both propose mechanisms where the place cells are themselves integral to the process of HV updating. In the model of Samsonovich and McNaughton (1997) the packet sends output to another multiple layered structure, which sends a return signal to an appropriately shifted location on the chart, thereby moving the packet. The model of Conklin and Eliasmith (2005) achieves packet updating using connections within the chart layer itself. Mittelstaedt (2000) notes that the HV representation being used by the rodent PI system seems mostly likely to be classifiable as geocentric Cartesian, on the basis that the head and place cells have responses of a form most easily interpreted as components of this type of system. Maurer (1998) takes a different approach to the MPI models and uses an egocentric polar representation of the HV to model hamster PI. Back propagation is used to train the network to update the HV based on the animal’s current movements. A closer correspondence to behavioural data is obtained using a modified version of the model which lacks the distance component of the HV.
Chapter 3

Methods

This chapter introduces the genetic algorithm and neural network methods used to construct PI models.

The General Network Genetic Algorithm (GeNGA) is introduced. This is the evolutionary algorithm used to evolve all the artificial neural network systems presented in this thesis and is a new GA, intended to allow the evolution of variable sized, variable topology networks without greatly constraining the type of neural network employed. The advantages of this approach are that two variable size and topology evolvable neural network types were deployed using exactly the same genetic algorithm (allowing a more direct comparison of the results), and that, because GeNGA can add and delete components from the networks as they are evolving, it was possible to employ a stochastic pruning process (see below) to automatically remove redundant components from evolved networks.

The three types of artificial neural network used in this thesis are introduced. The first type, the Continuous Time Recurrent Neural Network (CTRNN) is a relatively well studied, existing network type. CTRNN is a simple continuous time, mean firing rate model of biological neurons, and is in the process of being studied both analytically (Beer, 1995, 2006) and via simulation modelling (see for example Beer, 1996, 2003). The Modified CTRNN (ModCTRNN) and Continuous Time Recurrent System (CiTRuS) are both introduced for the first time in this work. ModCTRNN is an extension of CTRNN where weights are variables and as a consequence the network can more easily perform multiplication. This is the network used to evolve the most successful PI model (see Section 5.3.4). CiTRuS is a further extension of CTRNN which can perform multiplication easily (but by a more transparent mechanism than ModCTRNN) and also has variable time “constants”, allowing it to perform a memory-like operation I refer to as latching, in analogy with the digital latch. This model was used to manually construct a “vector navigation system” (see Chapter 7), as an extension of the evolved ModCTRNN PI network, which uses latching to remember the location of a food item in order to return there later.
Chapter 3. Methods

3.1 The General Network Genetic Algorithm

The simplest form of GA (Goldberg, 1989; Mitchell, 1996; Bück, 1996; Michalewicz, 1996) stores each genotype as a fixed length string of bits or floating point numbers, all of which are mutated using relatively simple, uniform operations. For the case of a bit string, all bits are subject to occasional bit flip operations, for floats all numbers would typically be constrained to the same range (e.g. $\in [0,1]$) and be mutated by the addition of a normally distributed random value. Recombination may also be used.

Various methods of extending this kind of fixed size encoding method have been tried, enabling the network to grow (and sometimes shrink) during evolution under the influence of new mutation operators which add or remove network components, for a review see Stanley (2004). Here the aim is not to investigate variable length encoding schemes in their own right, but to produce a tool sufficient for our purposes here, namely to evolve and compare more than one type of neural network on the same task, whilst allowing network components to be added and deleted during evolution. GeNGA fills this role in a minimal and direct way, and is not claimed to be the best or only way of achieving these aims. For a discussion of its limitations see Section 3.4.

GeNGA allows a network of heterogeneous components to be evolved. By ‘network’ we mean an abstract structure containing entities or nodes which are connected together in a certain way, similar to a ‘directed graph’ in mathematics. GeNGA network nodes need not correspond only to neural network nodes (neurons) in a simulated agent, but in principle could also specify parts of an evolvable morphology for instance.

3.1.1 Overview of GeNGA

GeNGA divides up a genotype into chunks called genes, which are the nodes of the GeNGA network. The term gene is only used for convenience, not due to any detailed similarity with biological genes. Genes are the indivisible units from which a genotype is built up, addition and deletion operators dealing only with whole genes. A gene contains a list of data items drawn from four classes: integers, floating point (real) values, ‘links’ and ‘targets’. Links and targets are used to specify and constrain the directed connections which exist between genes, allowing the creation of weights for a neural network or joints for a body morphology. The composition of a gene’s data items is defined by the class that the gene belongs to. As an example consider a CTRNN network encoded using GeNGA. We will assume the network consists only of neurons, sensors and weights. To encode such a network we define a gene class for each of these. The genotype will therefore contain genes from these three classes only. A neuron gene will contain the data required to define a neuron including its time constant and bias, a sensor will contain any parameter values associated with the sensor such as a gain value, and the weights will contain data defining the weight strength and the source and destination of the weight. The gene class defines all the details required to mutate a gene’s data and to apply addition or deletion operators where whole genes of the class are added or removed. Because of the complexity of performing most types of recombination between networks of differing topology, GeNGA is asexual (i.e. recombination is not used). GeNGA does not need modification in order to evolve a novel type of network, since all definitions of network data are given in a text configuration file. Due to GeNGA’s relative simplicity, there are some classes of network that it cannot deal with, for example it cannot easily
be made to enforce the constraint that the overall topology of the network must be feedforward.

3.1.2 Four data classes

Genes contain only four types of data: integer values, floating point values, links and targets. Their properties are explained below. In general each gene class definition contains a list of data item definitions, each of which names the data item, gives its type, valid range (if numerical) and names a mutation scheme to be applied to it. Links and targets are used to define references from one gene to another.

**Integer values**

The following is an example of a line from a GeNGA configuration file defining an integer data item:

```plaintext
int sensor_type 0 0 0 9 (randomise)
```

Here `int` defines the data item as an integer value, named `sensor_type` (names need only be unique within the gene class). The four following numbers define the valid range of the integer, the outer two values giving the valid range during normal mutation operations, the inner two giving the range within which the value is to be initialised when a new gene is created. `(randomise)` is the name of a parameter mutation scheme defined elsewhere in the file, giving various mutation rate parameters controlling mutation of the integer (see Section 3.1.4). Of the four range numbers, several are optional: if only two values are given GeNGA will take these as the total and initial range limits, if only one number is given GeNGA takes the integer to be a constant, which is not mutated and is simply initialised to the given value (this is a convenient way to define a gene class wide constant). If no mutation scheme is named after the range values the mutation scheme named `default` is used.

**Floating point values**

Here is an example of a line from a configuration file defining a floating point data item:

```plaintext
double weight -30 0 0 30 (default)
```

double defines the data item as a C double precision floating point value, named `weight`. Again the four numbers define the valid range, the outer two values giving the valid range during normal mutation operations, the inner two giving the range the value is initialised within when a new gene is created containing this value. `(default)` is the name of a parameter mutation scheme defined elsewhere in the file, defining further mutation related parameters (see Section 3.1.4). As with integer definitions, several parts are optional: if only two values are given these are taken as the total and initial range limits, if only one number is given the value is taken as a constant. If no mutation scheme is named the mutation scheme given `default` is used.

**Links**

Links are references to targets somewhere else within the same genotype. The link may be to a target in the same or a different gene. Link and target definitions define rules for which links can point to which targets. The following is an example defining a link data item:
link weight_source node_output sensor_output (default)

link weight_source defines the item as a link and names it. node_output sensor_output is a list (in this case of length two) of valid target types. This means that this link can only point to targets of type node_output or sensor_output. The name in parenthesis indicates the name of a link mutation scheme (see Section 3.1.4) defined elsewhere in the configuration file.

**Targets**

Targets are not true data items, since they contain no mutable data. Their only function is to provide targets for links to point to:

```target output node_output```

```target output``` defines and names the target. ```node_output``` gives the target type (a list of identifiers of any length can be given). No mutation scheme is given, since targets are not mutable data items.

### 3.1.3 Gene Class Definitions

A gene class definition contains three lines giving the name, gene mutation scheme and gene frequency constraints, plus a list of any number of data item definitions for integers, floats, links and targets. Here is an example:

```geneClass```

```name CTRNN_Node```

```mutation default```

```frequency 0 3 3 10```

```double time_constant 1 100 (log_scale)```

```double bias -30 0 0 30```

```target input node_input```

```target output node_output```

```</geneClass>```

CTRNN_Node is the gene class name, in this example we define a possible implementation of a CTRNN node. A gene mutation scheme (see Section 3.1.4) named default must be defined elsewhere in the configuration file, and gives parameters for use with the gene addition and deletion operators. The four frequency values are similar to the four range values for int and double data items: the inner two values define the minimum and maximum number of genes of this class that must be present in a genotype created randomly to initialise a new population, if omitted they are assumed equal to the outer values which define the minimum and maximum frequency of this gene type during all subsequent mutation operations. Addition and deletion of genes are not permitted to take the frequency outside of these limits. If the current frequency of a gene class in a genotype is found to be outside of these limits, it is assumed that the genotype has been hand coded to break the limits deliberately, or that the limits have been changed at some point during evolution.
Consequently the genotype is not discarded, but additions and deletions are only allowed to take the gene classes’ frequency towards the nearest valid frequency and not further from it.

3.1.4 Mutation Schemes
GeNGA configuration requires three types of mutation scheme (sets of mutation parameters) to be defined, one each for numerical data items, link data items and gene classes.

Parameter Mutation Schemes
Integer and floating point data items share the same type of mutation scheme, identified by tags marked parameterMutation. This set of parameters governs what can be called point mutations, operations affecting only the value of a single integer or floating point value within a gene:

<parameterMutation>
   name default
   probMutate 0.02
   probCreep 0.85
   creepStd 0.02
   probWrap 0.0
   scaleType linear
</parameterMutation>

The name (here default) is used to refer to the scheme in lines defining integer or double data items (the scheme named default is used if none is named explicitly). probMutate gives the probability that a data item using this scheme will be mutated in a given generation. If it is mutated, probCreep gives the probability that a creep mutation will be applied, otherwise the value is randomised with a uniform probability distribution across its entire valid range. For creep mutations the new value is the old value plus a normally distributed random value with a mean of zero and standard deviation equal to creepStd multiplied by the size of the item’s valid range. If the new value is outside of the valid range it is fixed with probability probWrap by wrapping around to the other boundary (this is suitable for circular variables), otherwise the value is truncated to the nearest boundary value. scaleType may be set to linear, log or reciprocal. If log or reciprocal are used, before a (creep or randomisation) mutation is applied the current value and the boundary values are mapped using the natural logarithm or reciprocal functions respectively, then the mutation is applied as per the linear (unmapped) case including the process of fixing out of range values, then the new value is mapped back to its normal range using the inverse of the original transforming function (i.e. the exponential and reciprocal functions respectively). Since the mutation operators never generate mutated values which are outside of the defined valid ranges, if the mutation operator is given a value which is already out of range before being mutated, it assumes the value was hand coded to this value for some reason. Therefore is does not apply the usual method of fixing an out of range value after a mutation has been applied, since this could result in the value jumping suddenly from the original out of range value to an in range value (even if a creep mutation is applied). Rather it applies a mutation in the normal manner, then accepts the mutated value if it is no further out of range than the starting value, but rejects it (reverting to the pre-mutation value) if it has become further out of range. In
this way out of range values can be mutated overtime (subject of course to selection) into in-range
values.

Doubles are mutated exactly as explained above. When mutating integer values the value is
first converted to the corresponding double precision floating point value, then a random value
between zero and one is added. This ensures that it is always possible for the integer to mutate to
a neighbouring value no matter how small the magnitude of creep mutation is. The value is then
treated in the same manner as doubles during mutation. After mutation the fractional part of the
value is discarded to give an integer.

Link Mutation Schemes

Link mutation schemes control link initialisations and mutations.

```xml
<linkMutation>
  name default
  probInitNull 0.0
  probFixNull 1.0
  probMutate 0.02
  probActiveNull 1.0
  probNullActive 1.0
</linkMutation>
```

The specified name is used to refer to the scheme in configuration file lines defining link data items
within gene classes. The basic link mutation operation is to randomly assign the link to a valid
target within the genotype. Every gene in the genotype is scanned for targets which have one or
more target type identifiers listed in the link’s list of valid target types. The link is then assigned
one of these targets at random. Since it is possible for there to be no valid targets available, we
must also have a link state, null, meaning that the link points to nothing. It may also be desirable
to allow links to be mutated to null even if targets are available. For example, imagine a neural
network being mutated. If we remove an incoming weight from a neuron we introduce a possible
disruption to the dynamics. If we then also in the same mutation operation reattach the weight to
another random neuron we may introduce a second disruption to the network. Hence removing
the weight without reattaching it seems likely to be a “smaller” mutation in some cases. There are
three types of link operation, all based on the basic random valid assignment operation: initialisa-
tion, mutation and repair. Links in a newly created gene, or in randomly generated first generation
genotypes are initialised to null with probability probInitNull, otherwise they are randomly
assigned a valid target if possible. Mutation is applied to a link with probability probMutate
each generation. Mutating links which are initially found null are set to a random valid target
with probability probNullActive and left null otherwise. Non-null mutating links are set to
null with probability probActiveNull and reassigned to a random valid target otherwise. Thus
it is possible to implement ‘small’ topology mutations if desired or conversely to completely avoid
generating null links with the appropriate link mutation scheme parameter values. The final link
operation is repair. If a gene is deleted there may be links remaining elsewhere in the genotype
which still point to targets within it. These must be repaired by either setting, with probability
probFixNull, to null or by a random valid assignment.
Gene Mutation Schemes

Gene mutation schemes control gene mutation, addition and deletion.

```xml
<geneMutation>
  name default
  allowAdd 1
  allowDelete 1
  allowMutation 1
</geneMutation>
```

The given name is used to refer to the mutation scheme from gene class definitions elsewhere in the configuration file. The three remaining options can be set to 0 or 1 depending on whether the corresponding operations (addition of a new gene, deletion of an existing gene, mutation of gene data items) are prohibited or allowed respectively.

3.1.5 Remaining GeNGA Configuration Options

A simple population structure is used based on the approach of evolution strategies (Bäck, 1996), whereby the evaluated genotypes are ranked by fitness, the best genotypes are retained as elites in the population and mutated copies of the elites are used to replace all other genotypes. Here is an outline of a complete GeNGA configuration file:

```xml
<ga>
  genotypes 20
  elites 3
  probResizeGenotype 0.1
  ratioAdd 1
  ratioDelete 2
  ...
  [MUTATION SCHEMES]...
  ...
  [GENE CLASS DEFINITIONS]...
</ga>
```

genotypes sets the population size (the total number of genotypes evaluated each generation), elites determines how many survive the selection phase. The elite genotypes are then used to regenerate the population: a mutated copy of the best elite is the first new genotype created, followed by a mutated copy of the second best, then the third and so on until the last elite, then the cycle starts again from the best elite. In this way all elites produce approximately the same number of offspring, until the population size limit is reached, in a fully deterministic fashion. The elites and their offspring form the population to be evaluated in the next generation. probResizeGenotype gives the probability that a single addition or deletion operation will be applied to a genotype each generation. If one is applied ratioAdd and ratioDelete determine the relative likelihood of each operation. If an addition is to take place the existing frequency of each gene class in the
genotype is counted, and the number of additional genes allowed for this class is found by subtracting the maximum allowed frequency from the current frequency. If no additional genes are allowed the addition operation is abandoned, otherwise a gene class is selected randomly, such that the probability of picking a class is equal to the proportion of available gene additions it accounts for. New genes are created by random initialisation of all constituent data items. Similarly for deletions, the number of available deletions for each gene class is determined by subtracting minimum allowed frequencies from the existing frequencies, then a gene is selected at random such that the probability is proportional to the number of genes available for deletion in each class. After a deletion the remaining genes are scanned for broken links pointing to targets within the deleted gene, which are fixed as described previously.

**Stochastic Pruning of Network Components**

To use GeNGA to prune redundant components from an evolved network the addition operator is disabled while the deletion operator remains active. All other mutation operators also remain active. Deletions which reduce fitness are removed from the population by selection as usual but those which do not may become fixed in the population. It should be noted that this process is likely to be better at removing redundancies than simple lesioning of a non-mutating network. This is because changes in network parameters under the influence of ‘point mutations’ can bring the network to a point where a deletion can be made which was not originally possible. Also, if a deletion occurs which does reduce fitness slightly, further compensatory mutations may occur to increase the fitness again overtime.

**Re-evaluation of Fitness**

Rather than store the fitness value obtained the first time a genotype is evaluated, all elites are re-evaluated each generation. This imposes a certain cost, since evaluations are the computationally expensive part of the simulation work carried out here, but prevents lucky genotypes with anomalously high fitness from becoming stuck in the population as elites. In the task considered for this thesis - PI mediated homing after a random initial excursion - there are effectively infinitely many initial excursions (defined by the location of a series of lights the agent must visit before homing), but we wish the network to allow accurate homing after any excursion, and also in the presence of sensory and actuator noise. This dictates that an agent be tested on a small number of excursions, and assigned a fitness based on the average performance. Each generation all genotypes are tested on the same set of excursions, allowing direct comparison of fitness values provided elites are also re-evaluated. The cost of reevaluation of elites is reduced by having a large number of mutants relative to elites each generation, since the fraction of mutants to the total population determines the degree of exploration of genotype space per unit of computational effort (i.e. per evaluation). One further factor to consider is the rate of drift through networks of neutral fitness in genotype space (Barnett, 2001), which might suggest in this case not having too many offspring per elite since, as soon as an offspring of equal or better fitness is found it should be used as a parent itself (rather than continuing to evaluate mutants of the current elite), since this would increase the rate of exploration of genotype space. Hence, considering only these factors with respect to the GA’s efficiency, we already have a trade off.
3.2 Neural Networks

Although obviously named in reference to biological neurons, in this thesis neural networks are employed not as realistic models of nervous systems, but as abstract dynamical systems capable of receiving sensory input from an agent in a simulated environment, of transforming and storing this information and of producing motor control signals. In addition to this they must also be suitable for construction by mutation and selection using a genetic algorithm. The CTRNN (Hopfield, 1984) has been well studied in this role as a control system for simple simulated agents (Beer, 2003), and as an abstract dynamical system in its own right (Beer, 1995, 2006). The CTRNN has been proved capable of approximating any smooth dynamical system (Funahashi and Nakamura, 1993), but this does not prove it is feasible in practise to evolve CTRNN models of all agent control problems we may be interested in, or that, if a given problem can be solved in this way, we can analyse and understand how the solution works. Beer’s choice of the CTRNN model as a subject of study is clearly partially motivated by its biological plausibility, as well as by its analytical tractability, but he makes no claims that it is an optimal agent or robot controller. Hence the study of variants seems not unreasonable. Indeed the CTRNN model has formed the basis of several novel ANNs in recent years including a homeostatic oscillator system (Di Paolo, 2002b,a) and a developmental controller system (Balaam, 2003), and is also commonly used with minor variations such as the use of a hyperbolic tangent function (Macinnes and Di Paolo, 2004) in place of the logistic activation function. Another reason to choose CTRNN is that it is guaranteed to be a non-divergent dynamical system, no matter what parameter values are used (provided all time constants are greater than zero) since each neuron has a negative feedback-type decay term and a sigmoidal output function. One of the aims of this thesis is to explore models closely related to CTRNN but which do not necessarily correspond so closely to biological neurons. The motivation being that it is often very hard to evolve a controller for an agent or robot, particularly for complex tasks. If we also impose the additional constraint of having the controller built from nodes modelling biological neurons we may make the task even harder. Of course biological models are a useful starting point since we know that nervous systems are capable of performing very complex operations indeed, and arose via an evolutionary process. One problem with this approach is that evolution may have exploited features of neurophysiology which are not yet represented by present day models, or which are not convenient to simulate. Izhikevich (2004) compares the computational costs of many spiking models against the number of neuronal phenomena they are capable of representing, and presents a new model aimed at representing the most for the least effort. This kind of approach seems logical if one has available a large amount of data regarding the brain area under study (in the case of Izhikevich the cortex). For the case of PI in desert ants (unlike PI in rodents) we have very little neurophysiological data (Labhart, 2000), and it seems premature to focus on biologically realistic models.

Once the idea of controlling the agent with a model of biological neurons is dropped, we are free to use operations not currently considered feasible with real neurons. This allows us to give individual neurons or nodes a greater functionality than in CTRNN, and in principle allows a smaller network to solve the same problem. This in turn may in practise make it easier for a GA to solve a given task, and also make it easier to understand how a given solution works since systems with a smaller number of variables are often easier to analyse.
We cannot of course then claim that any model of animal cognition constructed with non-realistic neurons is a possible specification for the real neural circuitry responsible for the behaviour, at least not with a simple one-to-one mapping between real and model ‘neurons’. The Neural Engineering Framework (NEF) (Eliasmith and Anderson, 2003) supposes that such a one-to-one mapping is essential for cognitive modelling, and provides a way to map any desired set of control equations into a system of biologically realistic neurons. This approach, whilst having the advantage of clear biological defensibility in this sense, must take as the starting point a set of control equations produced by the researcher. Hence it cannot be employed when the aim is to explore which control systems are possible and cannot give the same level of defensibility when the neurophysiological properties of the neurons in the animal under study are not known in detail (as is the case for PI in desert ants).

Our less biological control systems (ModCTRNN and CiTRuS) retain a key homology with real nervous systems, namely that they are coupled to the agent’s body and environment in the same way as the nervous system of the animal is coupled to it’s body and environment. The sensor input and motor output systems are intended to be justifiable as models of the situation for the ant, and the sensory input drives network dynamics which drives motor output which influences the agent’s behaviour.

Here two extensions of the CTRNN are introduced. Both are designed to be convenient evolvable agent controllers rather than models of biological neurons. The hope is that they may prove capable of evolving to solve more difficult tasks than the CTRNN, and that the resulting networks may prove easier to analyse and understand. Since both are supersets of the CTRNN model they retain the formal property of universal approximation. This also implies that, should we wish to hand build a network for a particular purpose, we can be sure that the two new models are at least as capable in this regard as CTRNN.

### 3.2.1 CTRNN

The CTRNN is based on a mean firing rate description of neural activity. This means that instead of modelling individual spikes, the rate of spike production is represented by a single number, the firing rate, which is normalised to the range $[0, 1]$. The internal state variable, $v_i$, of each neuron represents the membrane potential, each neuron being assumed to be isopotential (having the same potential across all of the cell body’s membrane). The internal state of neuron $i$, $v_i$, in a network of neurons, acts as a leaky integrator of incoming synaptic current, and its behaviour is governed by the following equation:

$$
\tau_i \dot{v}_i = -v_i + \sum_j w_{ij} \sigma(v_j + b_j)
$$

(3.1)

where $\sigma(x) = \frac{1}{1 + e^{-x}}$, $\tau_i$ is a positive time constant of neuron $i$, $w_{ij}$ is a real valued weight from neuron $j$ to neuron $i$ and $b_j$ is the real valued bias of neuron $j$.

The simplest network topology is full connectivity, where the subscript $j$ in Eqn. 3.1 runs over all neurons. This contains all possible less densely connected topologies as subsets, since setting a weight to zero has the same effect as removing the weight entirely. (Eqn. 3.1 can be used to describe a non-fully connected network if we define $\sum_j$ to iterate through all neurons $j$ which send
inputs to neuron $i$).

### 3.2.2 ModCTRNN

My original aim in producing the ModCTRNN model was to convert all CTRNN parameters into variables governed by leaky integrator equations similar to the CTRNN equation, thereby giving evolution the ability to control more of the network’s properties. This in practise meant converting only weights and time constants into variables, since conversion of the bias to a variable effectively turns it into a conventional CTRNN synaptic input, equivalent to a firing rate multiplied by a constant weight and hence does not provide any extra capabilities to a neuron. This aim is effectively achieved in the CiTRuS model, ModCTRNN achieves it only for weights. The original reason for making these changes was simply that they can give more functionality to the network for the same number of variables as discussed above. In particular note that the behaviour of any ModCTRNN weight is a function of input it receives from some subset of the whole network. This means that, since we know a (Mod)CTRNN can approximate any smooth dynamical system given sufficient neurons, we can control the behaviour of any weight according to any dynamical scheme we like. This can be viewed as a generalisation of the commonly used set of synaptic plasticity rules often applied to plastic neural networks, such as Hebbian learning. In this sense ModCTRNN can be viewed as CTRNN with a generalised synaptic plasticity scheme added.

It so happens that making weights variable is very similar to giving the network the ability to directly perform multiplications (in CiTRuS multiplication is implemented directly, rather than through having variable weights for reasons stated in that section). This means that the main extra ability of ModCTRNN over CTRNN is to perform multiplications in a relatively simple way. This makes ModCTRNN similar to a continuous time recurrent version of so-called sigma-pi networks, and in this sense it is not a completely new model since work exists evolving variable topology sigma-pi networks (Zhang, 1994, for example). Whilst not all authors consider multiplication to be possible in real neurons, there are several plausible mechanisms in the literature. See for example Bugmann (1992) for a spike coincidence detection method involving leaky integrator neurons with synaptic plasticity, and Häusser and Mel (2003) for one involving dendritic trees.

ModCTRNN weights are similar to CTRNN neurons, in that they can receive incoming connections, but they lack the ability to make outgoing connections, and also lack a sigmoid function. The state equation for a weight is as follows:

$$\alpha_i \dot{w}_i = -w_i + \beta_i + \sum_j w_j \sigma(v_k + b_k)$$  \hspace{1cm} (3.2)

where $w_i$ is the state of weight $i$ in a ModCTRNN network, $\sigma$ is defined as for CTRNN, $\alpha_i$ is a positive time constant, $\beta_i$ is a real bias term, $v_k$ and $b_k$ are as for the CTRNN (i.e. they are the internal state and bias of neuron $k$). We define $\sum_j$ to iterate through all weights $j$ which are inputs of weight $i$ and define $k$ to mean the neuron attached to weight $i$ via weight $j$. It should be noted that a fully connected network is no longer possible if we require that weights as well as neurons be fully connected since, when we connect two existing components together we create a weight to do so, which then becomes a new possible target for further incoming weights, and so on. This is because we allow “higher order” weights, in the sense of weights which input to
weights which input to weights etc. to any degree. It therefore follows that we cannot simply have a fully connected topology, and must use some kind of explicitly defined or evolvable topology.

The reason ModCTRNN allows multiplication to be performed easily is that, as is the case in CTRNN, weights act to multiply firing rates before they are inputted to receiving neurons as synaptic input. ModCTRNN weights are variables meaning that if, for example, two sensory inputs to a network need to be multiplied together, a weight can be set to follow the value of one of the inputs, and also be used as a weight from the second input to a receiving neuron. The input to the receiving neuron is then proportional to the product of the two sensor values. Note however that a weight can only take part on one such multiplication. It is possible to multiply more than two network variables together using chains of higher order weights (as in fact occurs in the evolved ModCTRNN PI system presented in Section 5.3.4).

The lack of a sigmoid function for ModCTRNN weights leads to a minor defect in the model. It is possible for a weight value to increase at a constant rate without limit, but only if the weight is part of a subset of the network which does not make any outward connections. For instance, a ModCTRNN weight which connects from a neuron, whose output is one, to itself effectively cancels out the negative feedback term $-w_i$, leaving only the bias and remaining input terms in the right hand side of Eqn. 3.2. Hence the weight can increase at a constant rate without limit. It is not sufficient to disallow such direct self connections, since a ring of connected weights can achieve the same behaviour. This feature turns out not to be a problem in practice, but the CiTRuS model was carefully defined to avoid this possibility altogether since it might allow numerical overflow related problems in simulation.

### 3.2.3 CiTRuS

CiTRuS has two main features which distinguish it from CTRNN (and ModCTRNN). Firstly each neuron has two sets of inputs instead of one. The first set acts as normal input as for CTRNN, the second inputs to a function whose output determines the decay or response rate (i.e. the reciprocal of the time constant) of the neuron's internal state. This means that the decay rate is no longer constant (as it is in CTRNN), and this allows a property I call latching. When the decay rate is high the neuron internal state is responsive to changes in synaptic input, but when it is low (or zero) the internal state does not change and acts as a memory. This is similar to the behaviour of a transparent latch in digital electronics. The synaptic input plays the role of the data input (D), the decay rate function input the role of the enable input (E) and the neuron output that of the data output (Q). The second distinguishing feature is the presence of multiplier nodes (m-nodes) in the network. An m-node is a node which simply outputs the product of all its inputs, and it therefore acts as a function not as a stateful network component. The basic topology of CiTRuS is for neurons to send outputs to m-nodes, which send outputs to neurons. Direct m-node to m-node connections are not allowed, and neither are direct neuron to neuron connections. Within these simple constraints any topology is allowed. A fully connected network of sorts is possible given a predefined number of neurons and m-nodes; every neuron receives input from every m-node and every m-node receives input from every neuron (but note that, because of the presence of $\phi$ weights, see Eqn. 3.7 below, this does not mean that all m-nodes output the same value). The m-nodes implement multiplication in a more direct way than ModCTRNN weight variables, and
avoid the constraints that ModCTRNN weights have, namely that one weight can only take part
in one multiplication operation, and that for two values to be multiplied one must be a weight and
one a neuron (or sensor), whereas in CiTRuS any network variable (neuron internal state) can take
part in any number of multiplications.

The CiTRuS equation is defined below, broken down into manageable pieces:

\[ \dot{v}_i = g_i(I_{ig}^i) [I_i - v_i] \]  

(3.3)

where \( v_i \) is the internal state of neuron \( i \), \( g_i \) is a function determining the decay rate taking \( I_{ig}^i \) as
its input and \( I_i \) is the ordinary input to the neuron. This equation is analogous to the CTRNN
equation (Eqn. 3.1) since it says that the internal state decays towards the value of the current
input value \( (I_i) \) at a certain rate. The similarity can be seem more easily if we write the equation as
\[ \frac{1}{\tau_{ig}^i} \dot{v} = -v + I_i. \]  

Now we see that \( \frac{1}{\tau_{ig}^i} \) is analogous to the CTRNN time constant \( \tau \). The reason for
having the equation expressed in terms of a decay rate instead of its reciprocal the time constant
of decay is that a decay rate of zero is allowable (the internal state simply remains constant at
its current value), but a decay rate of infinity is not since infinitely fast change cannot be easily
managed by a numerical integration engine. If we had \( \tau \) as a variable we would have to represent
the value of \( \tau = \infty \) to obtain a zero decay rate, and would have to prevent \( \tau \) from becoming zero.

The \( g() \) function is defined next:

\[ g_i(I_{ig}^i) = (a_i^g - b_i^g) \sigma \left[ c_i^g \left( I_{ig}^i + d_i^g \right) \right] + b_i^g \]  

(3.4)

where \( a_i^g, b_i^g \geq 0 \) are positive constants, \( c_i^g, d_i^g \) are real valued constants and the \( \sigma(x) \) function is as
defined for CTRNN \( \left( \frac{1}{1 + e^{-x}} \right) \). This function is a generalised sigmoid function, whose maximum
output is \( a_i^g \) and minimum output is \( b_i^g \). The output is therefore always positive or zero. This is
required since otherwise the internal state can diverge from the input value rather than converge,
and we will have the possibility of a divergent network with values going towards infinity. \( c_i^g \) and
\( d_i^g \) act to scale and shift the function’s input respectively. The \( g() \) function’s input \( I_{ig}^i \) is defined
next:

\[ I_{ig}^i = \sum_j w_{ig}^j m_j \]  

(3.5)

where \( w_{ig}^j \) is the real valued weight connecting from m-node \( j \) to neuron \( i \), and \( m_j \) is the output of
m-node \( j \). Likewise:

\[ I_i = \sum_j w_{ij} m_j \]  

(3.6)

where all terms are analogous to those of Eqn. 3.5. The output of each m-node is defined as:

\[ m_j = \prod_k \left[ \phi_{jk} (z_k - 1) + 1 \right] \]  

(3.7)

where \( \phi_{jk} \in [0, 1] \) is a weight like parameter associated with the connection from neuron \( k \) to m-
ode \( j \) and \( z_k \) is the output of neuron \( k \). Note that when \( \phi_{jk} = 0 \) we have \( \phi_{jk} (z_k - 1) + 1 = 1 \) and
when \( \phi_{jk} = 1 \) we get \( \phi_{jk} (z_k - 1) + 1 = z_k \). This arrangement is essential for a fully connected
network to function. Since in this case every m-node receives input from every neuron, every m-node output would be identical without the $\phi$ weights. With a subset of m-node input weights set to zero, the outputs from the corresponding neurons will not have any effect on the m-node product term. The output of each neuron is defined as:

$$z_i = (a_i - b_i) \sigma [c_i (v_i + d_i)] + b_i$$  \hspace{1cm} (3.8)

where $a_i, b_i, c_i, d_i$ are real valued constants ($a_i$ and $b_i$ can be negative unlike $a^g_i, b^g_i$). This again is a generalised sigmoid function which outputs values in the range $\in [a_i, b_i]$ regardless of the magnitude of the input value.

This defines CiTRuS for the case of a fully connected network with any number of neurons and m-modes. As for the fully connected CTRNN, this contains all less connected topologies as subsets where setting a $w$ or $\phi$ weight to zero is equivalent to deleting that connection altogether. For a less than fully connected network we simply define the summation and production terms to iterate through the subscripts of all incoming $w$ and $\phi$ weights respectively. Notice also that, for an evolvable topology network the $\phi$ weights also perform an important role. When a new $\phi$ weight is added to the network, $\phi$ can be set initially to a small or zero value, preventing the added weight from disrupting existing network functions. Later creep mutations to $\phi$ can bring the new weight into dynamic play gradually, in much the same way that slowly increasing a $w$ weight magnitude can achieve for a conventional network weight (see Section 3.4 for more about this feature).

### 3.3 Evolving CTRNN and ModCTRNN with GeNGA

This section details the way GeNGA was used to implement and evolve CTRNN and ModCTRNN networks. CiTRuS could also be handled by GeNGA, but was only in fact used to make hand designed models during this thesis, since its design was only finalised towards the end of the project.

The following mutation parameter settings are referred to in the following sections (the parameter mutation scheme named circular is not used for mutating the network components defined below, but is shown to illustrate how parameters on a circular scale are mutated, as is needed to encode an angular parameter):

```xml
<parameterMutation>
  name       default
  probMutate 0.02
  probCreep  0.85
  creepSD    0.02
  probWrap   0.0
  scaleType  linear
</parameterMutation>

<parameterMutation>
  name       circular
  probMutate 0.02
</parameterMutation>
```
probCreep 0.85
creepSD 0.02
probWrap 1.0
scaleType linear
</parameterMutation>

<parameterMutation>
  name log_scale
  probMutate 0.02
  probCreep 0.85
  creepSD 0.02
  probWrap 0.0
  scaleType log
</parameterMutation>

<parameterMutation>
  name randomise
  probMutate 0.02
  probCreep 0.0
  creepSD 0.0
  probWrap 0.0
  scaleType linear
</parameterMutation>

<linkMutation>
  name default
  probInitNull 0.0
  probFixNull 0.0
  probMutate 0.05
  probActiveNull 0.0
  probNullActive 1.0
</linkMutation>

<geneMutation>
  name default
  allowAdd 1
  allowDelete 1
  allowMutation 1
</geneMutation>
3.3.1 CTRNN with GeNGA

Here we define the configuration of GeNGA CTRNN neurons and weights. This is enough to show how a CTRNN is encoded and evolved using GeNGA. The definition of the additional sensors and motors required to construct a PI agent is left to the corresponding chapters dealing with these experiments.

**Neurons**

Neurons encode a time constant and bias, and also have a target for incoming and outgoing weights to connect to. Biases are initialised to zero to give the centre-crossing property (Mathayomchan and Beer, 2002) whereby the bias is equal to negative one-half of the sum of all incoming weights. Since new weights are also set to zero (both when a new genotype is created and when a single new weight is added later during evolution) new neurons have no non-zeroed incoming weights and hence a bias of zero is centre crossing - albeit in the most trivial sense. This simple method of centre crossing has the drawback that new neurons, beginning with no incoming or outgoing weights, are not under selective pressure to begin with and hence the bias may drift to a large value, and hence could cause saturation of the neuron either on or off before any non-zero weights are attached.

The GeNGA configuration for CTRNN neurons is shown below (please see Section 3.3 for the mutation settings referred to in the following `geneclass` definitions):

```xml
<geneclass>
  name CTRNNNeuron
  mutation default
  frequency 0 10

  float v0 -50 0 0 50
  float bias -50 0 0 50
  float tau 0.01 1000 (log_scale)

  target inter_in neuron_in
  target inter_out neuron_out
</geneclass>
```

**Weights**

Weight genes encode a single weight parameter, initialised to zero for new weights. They also contain two links pointing to the sending sensor or neuron and the receiving neuron. The incoming link points to a sensor or neuron output target, the outgoing link points to a neuron input target. By initialising a weight to zero we ensure it is always phenotypically neutral and non-disruptive to inserted a new weight into an existing network. Later creep mutations may then bring it into play gradually, or randomisation mutations may cause large changes in its value. The fact that weights are encoded as separate genes allows them to be stochastically pruned one at a time as described in the GeNGA section above. The mutation settings for the incoming and outgoing links are set to prevent any new weights from being initialised with null (unconnected) links. Where, as in Chapter 5, networks were constrained to be bilaterally symmetrical (by decoding every encoded
network gene as two bilaterally arranged network components), weight genes also encoded an integer value defining whether the weight was ipsi- or contralateral. Contralateral weights were decoded to connect from an ipsilateral source to a contralateral target, whereas ipsilateral weights connected from an ipsilateral source to an ipsilateral target.

The GeNGA configuration for CTRNN weights is shown below along with the mutation settings:

```xml
<geneclass>
  name CTRNNWeight
  mutation default
  frequency 0 30

  float weight -100 0 0 100
  float alpha 0.01 1000 (log_scale)

  link link_in neuron_out sensor
  link link_out neuron_in
  int contralateral 0 1 (randomise)
</geneclass>
```

In Chapter 4, dealing with a simplified one dimensional analogy of PI where the agent measures the distance travelled on a straight line, a different encoding scheme was used, where each neuron is encoded with ten incoming weights as part of the same gene. This had the disadvantage of not allowing the stochastic pruning of redundant weights.

### 3.3.2 ModCTRNN with GeNGA

The encoding used for neurons is identical to CTRNN but weights are more complex. They contain (see Eqn. 3.2) a time constant $\alpha_i$, a bias $\beta_i$, a target for incoming weights (which modify the weight's own value), an outgoing link to neurons or weights and an incoming link from sensors or neurons. Weight genes could also encode the initial weight value (i.e. $w_i(t = 0)$), but instead it is more convenient to initialise a weight to its bias value, such that weights with no incoming connections remain constant throughout the simulation. This reduces the effective dimensionality of the network for analysis. The bias of newly created genes is set to zero, so that new weights will be of magnitude zero and hence neutral when first inserted as was the case for CTRNN weights.

```xml
<geneclass>
  name ModCTRNNWeight
  mutation default
  frequency 0 30

  target mod_weight weight
  float bias -100 0 0 100
  float alpha 0.01 1000 (log_scale)
</geneclass>
```
3.4 A More Principled Approach to Evolving Variable Topology Networks

The ideas discussed in the remainder of this chapter were not implemented or used for the models presented in later chapters. The following section explores the limitations of the methodologies that were used, and suggests more advanced alternatives.

3.4.1 Custom Mutation Operators

GeNGA adds and removes network components in a rather crude way and was not designed to allow the later addition of custom mutation operators of any kind to tailor it towards any particular neural network type. The only available refinements are the ability to prevent connections being formed which are obviously nonsensical from the immediate local context (such as attempting to send input to a sensor), and the ability to specify that parameter values should be initialised within a different range to that within which they will later be mutated (such as creating weight and bias parameters set to zero rather than large random values). Components are also only ever added one at a time, there being no way to specify that a group of components can be added in a certain predefined configuration. The design of ModCTRNN is better understood in the light of this fact. Since only one component can be added at a time, it is probably better to have that component able to perform a useful task immediately, by itself. A ModCTRNN weight can, without any further topological mutations, act as a new input to an existing network component (i.e. to either a weight or neuron), and can later be mutated to become a fully fledged network variable by the addition of inputs to the new weight itself. In CiTRuS, the equivalent of a ModCTRNN weight is a set of no less than five components: a neuron, two $\phi$ weights, an m-node and a w weight (where the neuron is the equivalent of the state of the ModCTRNN weight, the m-node and $\phi$ weights act to multiply an existing neuron output by the output of the new neuron, and the w weight inputs this value into another existing neuron). Such a structure could of course be built up by chance from five separate mutations, but, since each mutant must be evaluated, this appears to impose a greater computational cost to achieve the equivalent of a simple weight addition in ModCTRNN. It seems likely that a more efficient GA could be constructed if we allowed mutations to be customised to the type of network being mutated by allowing, for instance, more complicated topological mutations.

We could also allow global network properties to influence mutations, and prevent useless junk components from building up. For example, consider that there is nothing to stop GeNGA from creating new components which are topologically disconnected from any of the network’s outputs, and hence cannot possible influence the phenotype. To prevent this we need more than local rules to constrain the topology. Ideally we would prevent any mutation from creating phenotypically useless components, by analysing the existing topology before altering it. Junk components, unless filtered out before fitness evaluation, increase the load on the numerical integration engine and
slow the agent simulator. Mutations applied exclusively to junk components are also inefficient, since the mutant, although requiring evaluation, will not differ from its parent. We now introduce a framework for designing efficient mutation operators based on the idea of creating and exploiting neutral network in the fitness landscape.

3.4.2 Exploitation of Neutral Networks

When a population of genotypes is evolved by a genetic algorithm under the influence of mutation and selection over a significant number of generations, often the fitness of the population will appear to get stuck at a particular value for extended periods. This means that the GA is unable to find any fitter mutant during this period. After a time a fitter mutant may be discovered and the fitness value may increase for a short time until another fitness plateau is reached. There are two main classes of explanation for this phenomenon: that the period of stable fitness is caused by the population being stuck on a fitness peak which it escapes from by jumping to a higher peak, or that the population is drifting at random through a ‘neutral network’ of equally fit genotypes which it exits by finding a ‘portal genotype’ which is a route to regions of higher fitness (van Nimwegen and Crutchfield, 2000). Note that, in contrast to the former mechanism, the portal genotype is only separated from the neutral network by an ordinary, small mutation, not by a big jump, nor by a sequence of genotypes with a fitness lower than that of the network. Roughly speaking, a neutral network is defined as a network of genotypes all sharing the same fitness and all connected together by neutral mutation operations (a neutral mutation being one which changes the genotype without changing its fitness). Simulations suggest (ibid.) that a population would find it easier to discover genotypes of higher fitness by the neutral network mechanism than by the other. Consequently, if we believe neutral networks of genotypes with equal fitness are likely to exist in the fitness landscape we are exploring, it is logical to design a GA to be efficient at searching them. Barnett (2001) suggests the Netcrawler algorithm as a potentially optimal method of searching under these conditions, which is designed to optimise the rate of drift through the neutral networks, and hence maximise that rate at which a neutral network is explored for portal genotypes. Netcrawler is based on the assumption that a genotype’s fitness can be determined with complete accuracy for the cost of a single evaluation, and that therefore neutral mutants can be easily identified. This assumption is not met by the PI simulation work presented here, due to the presence of noise, and so we can no longer simply use this algorithm. We can however make use of the idea of neutral networks in the design of our mutation operators and network models.

Our networks have an evolvable size and topology. The mutation operators must therefore include those which add and delete weights and neurons, as well as those which change parameter values. This brings a number of new problems to the design process when compared to fully connected networks. Stanley (2004) introduces an algorithm with similar aims called NEAT, which employs custom mutation operators to add new neurons to a network. NEAT grows networks by deleting an existing weight and replacing it by a weight-neuron-weight group. The method used involves giving the new mutant network a temporary fitness bonus to offset the expected decrease in fitness resulting from inserting a new neuron. NEAT allows recombination between networks with different topologies - a difficult problem to solve algorithmically since it is not clear which parts of the network should be considered homologous and suitable for genetic exchange. NEAT
lacks deletion operators, and so cannot be used to automatically prune evolved networks. The basic approach of NEAT, the application of specially designed mutation operators to mutate network topology, taking into account the existing, global topology, and to ensure minimal disruption of existing functions, is felt to be sound, and is developed in the following sections.

3.4.3 Smooth Mutations

The approach to topology mutations considered here is to render the operation of addition of network components phenotypically neutral where possible, and to apply subsequent creep mutations to associated parameters in order to bring the new components into dynamic play gradually. With respect to deletion this idea works in reverse, so that before we delete a component we attempt to apply creep mutations to make its removal neutral. This is an extension of the idea of a neutral mutation which I call smooth mutation. The key concept is that, taken to the limit of infinitesimally small creep mutations to parameter values, there are no discontinuities in network behaviour at the moment of addition or deletion of new components (neurons and weights), hence we can create neutral networks which span multiple network topologies. Bifurcations will still give rise to sudden changes in network dynamics, but this is unavoidable and desirable for exploratory purposes. The most complete set of mutation operators would include several or all available methods of changing the network topology that could be achieved in a neutral or gradual way. This could even include topology mutations which rearranged existing components rather than added or deleted anything. As well as designing mutation operators to this end, the design of the neural network itself could be partly based on the idea of creating and exploiting neutral networks. Hence an ideal dynamical system agent controller would not only be a powerful dynamical approximator, but also have a readily evolvable topology.

There are at least three different ways to introduce a new variable into a dynamical system gradually:

(i) a function which becomes a variable - a neuron with a much faster time constant than anything else in the system can be, in certain cases, at equilibrium. The neuron therefore acts more as a function than a variable since its state, while not constant, is a function of its inputs alone. If the function performed is the identity function, such as by passing its input through the linear part of its sigmoid function, then inserting such a neuron anywhere in the network can be neutral, and subsequent changes to its parameters can bring it gradually into play (selection permitting).

(ii) a parameter which becomes a variable - a neuron which does not change its value at all, because of a large time constant, or because its inputs have a small magnitude, is acting more like a parameter than a variable with respect to the rest of the network. If this parameter acts in a neutral way, such as contributing zero to a summation, or one to a production term, it can be inserted neutrally, and subsequent changes to its parameters can gradually bring it into play.

(iii) duplication - if a neuron is duplicated into two new neurons, each identical to the original save that their output weights be halved, then (in the absence of noise) the change will be neutral, but subsequent small changes to their parameters can effect changes which were impossible before the duplication.

The reverse of these three operations can in principle bring about a neutral deletion. The operation of a smooth mutation system can be summarised by three main ideas:
(i) phenotypic neutrality - the idea that all or most changes to network size and topology should be neutral or almost neutral with respect to the phenotype. Since network components are inherently discrete objects, there is a danger that addition or deletion will generate large, discontinuous changes to network behaviour unless we take special care to ensure otherwise.

(ii) topological effectiveness - the idea that, although neutral, the change should be a potentially significant one given subsequent changes to parameters but without necessarily requiring any further changes to topology. This excludes, for example, adding a new neuron which is topologically disconnected from everything else since selection cannot possibly act on its parameters, but allows a new neuron which is connected to existing ones by zero weights since no further topological changes are required to bring it into play.

(iii) mutational poise - parameter values of newly added components should be initialised carefully, such as by using centre crossing for all sigmoid functions (Mathayomchan and Beer, 2002), so as to allow future creep mutations to bring new components into dynamic play easily but gradually. If a new neuron is saturated fully on or off, subsequent small changes to its incoming weights will have little or not effect.

Example Addition Operators

Taking the example of the addition of weights, if we initialise all newly added weights to zero they are guaranteed to have no immediate phenotypic effect. If we mutate weights using creep mutations (i.e. by adding random values from a gaussian distribution with a small standard deviation) the weight is likely to move away from zero by small steps over several generations. Provided the magnitude of mutations is small relative to existing weights in the network, the new weight is likely to come into play gradually.

To add a neuron smoothly, we can obviously connect it to existing neurons by zeroed weights, using the parameter-becoming-a-variable method. To insert a neuron smoothly inbetween two existing neurons we can use the function-becoming-a-variable method. In the same way that NEAT adds neurons, we initially delete (taking the example of CTRNN) a weight and replace it with a weight-neuron-weight chain. We set the neuron’s time constant to be as small as possible, scale down its input onto the linear part of its sigmoid, then scale the output back up again, hence creating an identity function provided the time scale of variations in its input is longer than its time constant, otherwise it will act as a temporal filter. We ensure the overall effect of the chain is identical to that of the deleted weight. The duplication method can also be used.

Example Deletion Operators

If we play these processes in reverse, we can envision a neutral deletion operator. But we need to do more work to encourage the network to move towards a configuration where a weight or neuron can be neutrally deleted, and of course during evolution selection will prevent certain such changes from taking place if the component concerned performs a vital function. We imagine that a certain network weight is in fact a redundant component, but that immediate removal will reduce the fitness since other parts of the network have evolved to depend on its presence opportunistically. We now assume that each weight has a meta-mutational flag associated with it. The value of the flag is occasionally flipped by mutation. When active the flag modifies the action of creep mutation such that the value of the weight may only decrease in magnitude. When any flagged weight is mutated such that its sign is changed (i.e. such that the value passes through zero), the weight is
deleted. The rationale is that it will take several generations for the weight to reach zero, allowing time for compensatory mutations to occur elsewhere, making the weight’s removal more neutral. Of course, should a non-redundant weight become flagged like this selection will probably block its mutation and deletion until the flag is once again deactivated. This mechanism actively ratchets weight values towards zero, where normal mutations might only cause them to drift randomly.

Neutral deletion of neurons can easily be accomplished by removing any that have had all their outgoing weights deleted in this way. Also, we could devise ratchet mechanisms which tended to move neurons towards an identity function or a parameter, or that moved pairs of neurons towards being dynamically identical.

3.4.4 Final Remarks

It should be emphasised that we do not want to force all mutational events to be neutral or small. There may be times when a few large mutations are more likely to take the population to a higher fitness than many small changes. If, however, we consider the incremental construction of a large, complicated network then \textit{a priori} we expect the phenotypic effect of each component to be intimately linked to many other components. Consequently changes in network structure will probably sometimes require coordinated change across many components and parameters, which are unlikely to happen all at once in one lucky mutant. Therefore we are more likely to improve fitness or facilitate diffusion over neutral networks by making many smaller changes over several generations.
Chapter 4

One Dimensional Path Integration

4.1 Overview

A simplified analogy of PI is presented involving measuring distances travelled in a one-dimensional space. GeNGA is used to evolve CTRNN controllers capable of performing the task. The agent has a food and a speed sensor. Output neurons specify forward and reverse thrust. A final output neuron signals that the agent thinks it has reached its final destination. An externally imposed random “wind” force acts to perturb the agent’s motion. Two variations of the task are studied. In the first, the agent must move forward until the food sensor is activated, when it must return to its starting location and activate the signal neuron. In the second the agent again moves forward until the food sensor is activated, when it must now travel forward for an equal distance before activating its signal neuron. Both tasks are solved successfully by GeNGA/CTRNN. Analysis of the controllers is used to identify the principles they use. The two tasks are found to be solved using different strategies, but both involve the use of a neuron with a large time constant, acting as an integrator.

4.2 Methods

The agent was controlled by a CTRNN network (see Section 3.2.1) evolved using GeNGA (Section 3.1). The CTRNN equation was modified by the inclusion of a gain parameter, but otherwise identical to that defined above. The gain parameter modifies the CTRNN equation (Eqn. 3.1) to:

$$\tau_i \dot{v}_i = -v_i + \sum_j w_{ij} \sigma[g_j v_j + b_j]$$

Unlike the later two dimensional PI modelling work, the network here was encoded in such a way that individual weights could not be stochastically pruned. Rather each neuron had a fixed number (10) of incoming weights associated with it. This encoding scheme therefore allowed only pruning of whole neurons. It was found that only the three output neurons were required to solve the task. Hence no automatic stochastic pruning was available since all neurons were required. Each neuron was encoded as a set of five parameters ($v_0, b, \tau, g, f$) plus ten weights ($w$), and ten GeNGA links (see Section 3.1.2) defining which neurons or sensors the weights were inputting from. $\tau$ and $g$, \ldots
the time constant and gain parameters, where encoded and mutated in $\log_{10}$ form. $v_0$ and $b$ are the initial state and bias parameter respectively. $f$, an integer value $\in \{0,1\}$, is a centre-crossing flag, which, if set, caused the encoded bias and initial state parameters to be ignored and the centre-crossing (Mathayomchan and Beer, 2002) values $v_0 = \frac{\sum w}{2}$ and $b = -g\frac{\sum w}{2}$ to be used in their place. Each genotype also encoded $\theta$, the threshold for the signal neuron. Each numerical parameter was mutated each generation with probability 0.15. A random value was added to mutating parameters drawn from a normal distribution with zero mean and standard deviation equal to 0.1 times the maximum valid range of the parameter. Out of bounds values were fixed by truncation to the exceeded boundary. The valid ranges for the parameters were: $v_0 \pm 200$, $b \pm 20000$, $\log_{10} g \pm 2$, $\log_{10} \tau \in [0,4]$, $f \in \{0,1\}$, $w \pm 20$ and $\theta \in [0,1]$. A population of 15 genotypes was used. The best 5 were copied, mutated and used to replace the worst 5 each generation. One fitness evaluation consisted of 5 trials, one trial being a run of the task with all random features and processes resampled, the fitness assigned to the genotype was the worst of the 5 trials. Five separate GA runs were performed for each experiment.

The agent’s motion in the one dimensional space was governed by simple Newtonian physics, assuming the body was a particle influenced by the forces generated by forward and reverse thrusters and by the external wind force. The agent’s spatial location, $x$, was governed by the following equation:

$$\frac{d^2 x}{dt^2} = \frac{F}{M_{agent} + M_{food}}$$

where $F$ is the net force applied to the agent, $M_{agent}$ is the agent’s mass (set to 1) and $M_{food}$ is the mass of a food item carried by the agent during the second leg of the journey after the food sensor has been activated (set to 0 during the first leg, set to 1 during the second leg). The net force $F$ is calculated as follows:

$$F = F_{max} (\sigma_F - \sigma_R) - C_{drag} \left( \frac{dx}{dt} - w \right)$$

where $F_{max}$ is the maximum thruster force (set to 1), $\sigma_F, \sigma_R$ are the outputs of the neurons controlling forward and reverse thruster activation respectively, $C_{drag}$ is the linear drag coefficient (set to 1), $\frac{dx}{dt}$ is the agent’s current speed and $w$ is the current wind speed. Wind speed was set to a value uniformly randomly drawn from the interval $[-1,1]$ for a time interval uniformly randomly drawn from the interval $[0.2,20]$. After this time had elapsed a new random wind speed and duration were selected using the same process, and so on. The agent had a signal output neuron with an evolvable threshold parameter. The signal was considered to have been activated the first time the output of the neuron changed from a sub-threshold to a super-threshold value. Hence, if the neuron started the trial above threshold, the value must first go below threshold, then increase again to above threshold before a signal was registered. The system was numerically integrated using the Euler first order method, using a step size of 0.2.

The agent had two sensors. The speed sensor signalled the agent’s current speed on a normalised scale where 1 indicated the maximum possible forward speed and 0 the maximum reverse speed. Given the parameter values used the maximum forward and reverse speeds (with maximum wind assistance) were 2 and $-2$ respectively. The food sensor outputted a value of 0 from the be-
ginning of the journey, but changed to outputting 1 after the “food” location had been reached until the end of the trial.

4.2.1 The Two Tasks

The two tasks were: (i) the two-way task, where the agent was required to move forward for a random distance, whereupon its food sensor was activated and its mass increased from $M_{\text{agent}}$ by $M_{\text{food}}$; the agent was then required to return to the place it started the trial and activate its signal neuron. (ii) the one-way task, where the agent was required to move forward to the randomly selected food location exactly as before, but was then required to continue travelling forward for a distance equal to that which it had already travelled before activating its signal neuron. The location the agent was required to signal at is referred to below as either the goal or nest location. The time given for the agent to complete the task was 2.5 times the time needed to complete both legs of the trip travelling at maximum speed with a wind speed of zero. Agents were evolved to solve one or other of these tasks (not both). The interval in which the food location was randomly placed was varied during evolutionary runs, beginning small and expanding gradually as fitness improved. The placement zone began at $[5, 10]$ distance units. Once high fitness solutions were evolved the size was successively doubled (waiting each time until fitness had recovered and stabilised after the change) until the zone was $[5, 400]$. For the five trials used to evaluate each genotype each generation, the food location was set using a stratified random sampling scheme where the total placement zone was divided into five equal sized sampling blocks, one for each trial.

4.2.2 The Fitness Function

The fitness function utilised several values collected during the trial. These values were the nearest $n_{\text{food}}$ and furthest $f_{\text{food}}$ distance of the agent from the food location during the first leg, the time taken to reach the food $t_{\text{food}}$, the nearest $n_{\text{nest}}$ and furthest $f_{\text{nest}}$ distance of the agent from the nest during the return leg, the time taken to reach the nest $t_{\text{nest}}$, the lowest $l_{\text{sig}}$ and highest $h_{\text{sig}}$ value attained by the signal neuron output, the status of the trial when the agent signalled and finally the distance $d_{\text{sig}}$ of the agent from the food or nest when it signalled (the former if it signalled before reaching the food and the latter if it signalled after).

The final fitness value for a trial was the sum of three values $F_1, F_2, F_3$, reflecting the agent’s performance at reaching the food location, then reaching the correct goal location and accurately signalling the goal location respectively.

**Reaching the Food**

If the agent didn’t reach the food $F_1 = 0.01(a_1 - a_2)$ where:

$$a_1 = 1 - \frac{n_{\text{food}}}{d_{\text{init}}}$$

$$a_2 = \left[ \frac{f_{\text{food}} - d_{\text{init}}}{d_{\text{max}} - d_{\text{init}}} \right]_{[0,1]}$$

where $d_{\text{init}}, d_{\text{max}}$ were the agent’s initial and maximum possible distance from the food and the subscript indicates the value is clipped to the interval $[0, 1]$. $a_1$ rewards small food-nearest-approach
values \( n_{food} \). \( a_2 \) penalises large food-furthest-distance values \( f_{food} \).

If the agent reached the food \( F_1 = 0.01a_3 + 0.01 \) where:

\[
a_3 = \left\lfloor \frac{\text{t}_{\text{min}}}{\text{t}_{\text{food}}} \right\rfloor \leq 1
\]

where \( \text{t}_{\text{min}} \) is the minimum possible time to reach the food for this trial, and the subscript indicates the value is clipped to be less than or equal to one. \( a_3 \) rewards small time-to-food values \( (t_{food}) \).

**Reaching the Goal Location**

If the agent reached the food but not the nest \( F_2 = 0.01(a_4 - a_5) \) where:

\[
a_4 = \left\lfloor 1 - \frac{n_{\text{nest}}}{d_{\text{init}}} \right\rfloor \geq 0
\]

where \( d_{\text{init}} \) was the agent’s initial distance from the nest at time of reaching the food. \( a_4 \) rewards small nest-nearest-approach values \( (n_{\text{nest}}) \).

If the agent reached the nest after reaching the food \( F_2 = 0.01a_5 + 0.01 \) where:

\[
a_5 = \left\lfloor \frac{\text{t}_{\text{min}}}{\text{t}_{\text{nest}}} \right\rfloor \leq 1
\]

where \( \text{t}_{\text{min}} \) was the minimum possible time the agent could have taken to complete the task. \( a_5 \) rewards short time-to-nest values \( (t_{nest}) \).

**Signalling Accuracy**

If the agent didn’t signal during the trial and the lowest signal neuron output value \( (l_{\text{sig}}) \) was greater than the signal threshold \( (\theta) \) \( F_3 = 0.01a_6 \) where:

\[
a_6 = 1 - l_{\text{sig}} - \theta
\]

\( a_6 \) rewards smaller lowest-signal-output values.

If the highest signal neuron output \( (h_{\text{sig}}) \) was less than the signal threshold \( (\theta) \) then \( F_3 = 0.01a_7 \) where:

\[
a_7 = 1 - \theta + h_{\text{sig}}
\]

\( a_7 \) rewards larger highest-signal-output values.

If the agent did signal during the trial, but signalled before reaching the food then \( F_3 = 0.01a_8 + 0.01 \) where:

\[
a_8 = \left\lfloor 1 - \frac{d_{\text{sig}}}{d_{max}} \right\rfloor \leq 1
\]

where \( d_{\text{sig}} \) was the distance from the food when the agent signalled and \( d_{max} \) was the maximum possible distance the agent could be from the food during the trial. \( a_8 \) rewards signalling closer to the food.

If the agent signalled after reaching the food then \( F_3 = 0.94a_9 + 0.02 \) where:
\[ a_0 = \left[ 1 - \frac{d_{\text{sig}}}{d_{\text{max}}} \right]_{\leq 1} \]

where \( d_{\text{sig}} \) was the distance from the nest when the agent signalled and \( d_{\text{max}} \) was the maximum possible distance the agent could be from the nest during the trial. \( a_0 \) rewards signalling closer to the nest (or goal) location. The final fitness was \( F_1 + F_2 + F_3 \).

Runs were performed where the number of neurons was not allowed to evolve, but was fixed at the three output neurons only. Runs were also performed where the number was allowed to evolve but this did not produce any improvement over the three neuron networks and these results will not be presented here.

### 4.3 Results

#### 4.3.1 The Two-way Task

For the two-way task, all five runs evolved agents able to successfully complete the task. Initial comparisons between the best agent in each population suggested that all used approximately the same strategy, so analysis was concentrated on the best agent from the five runs. Fig. 4.1 shows the CTRNN network, Table 4.1 shows the parameter values, Fig. 4.2 shows the dynamics of the network. The following paragraph presents a simple analysis of the main features of the evolved network, able to explain the agent’s ability to signal at approximately the correct location. The later paragraphs deal with the network in more detail. A series of modifications is made to the network to determine whether all weights have a function regarding the agent’s task or whether some are redundant.

The food sensor acted to reverse the agent by disabling the forward motor neuron and activating the reverse motor neuron. Before reaching the food the forward motor neuron was approximately saturated on and the reverse neuron off, after reaching the food the situation rapidly reversed. The state of the signal neuron was roughly linear with the position of the agent, indicating it was approximately integrating the output of the speed sensor. Since the speed sensor outputs positive values for both forward and reverse speeds, an integrator cannot function by simply integrating these values. In order for backwards travel to have an equal and opposite effect on integrator state the integrator neuron must have an additional constant input acting as an offset, such that speed sensor output plus offset is zero when the agent is stationary, positive when it moves forward and negative when it moves backwards. There is no spare neuron available in the network to provide such a constant output (and the neuron cannot use its bias, since this affects only the mapping from its internal state to its output). Instead the network uses the combined outputs of the forward and reverse motor neurons. Since one of these is always active (except for an instant when the agent is changing direction), but never both at the same time, and since both are always either saturated on, or saturated off, the sum of both outputs is roughly unity. The integrator weights from these neurons are roughly the same (11.3 and 10.8) and roughly negative one half times that of the weight from the speed sensor (-19.3). This gives the integrator neuron its offset. Had the speed sensor output been defined in the range \([-1, 1]\) this trick would not have been needed.

This explanation appears sufficient to explain the functionality of the agent, but does not ex-
Table 4.1: Parameter values of the evolved CTRNN network performing the two-way distance measuring task. Fo, food sensor; Sp, speed sensor; Fw, forward motor neuron; Rv, reverse motor neuron; Sg, signal neuron. \( w_{X,Y} \) indicates a weight from component X to component Y. \( \theta \) is the signalling threshold of the signal neuron. All neurons have the centre-crossing flag \( f \) set. Hence the values shown for \( v_0 \) and \( b \) are the centre-crossing ones. See Fig. 4.1.

<table>
<thead>
<tr>
<th>Component</th>
<th>Parameter Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w_{Sp,Sg} )</td>
<td>( w = -19.317 )</td>
</tr>
<tr>
<td>( w_{Fo,Fw} )</td>
<td>( w = -38.166 )</td>
</tr>
<tr>
<td>( w_{Fo,Rv} )</td>
<td>( w = 25.653 )</td>
</tr>
<tr>
<td>( w_{Fw,Sg} )</td>
<td>( w = 11.302 )</td>
</tr>
<tr>
<td>( w_{Fw,Rv} )</td>
<td>( w = -33.303 )</td>
</tr>
<tr>
<td>( w_{Rv,Sg} )</td>
<td>( w = 10.781 )</td>
</tr>
<tr>
<td>( w_{Rv,Fw} )</td>
<td>( w = 10.321 )</td>
</tr>
<tr>
<td>( w_{Sg,Fw} )</td>
<td>( w = 12.603 )</td>
</tr>
<tr>
<td>( Fw )</td>
<td>( v_0 = -7.621, b = 3.521, g = 0.462, \tau = 1.1246, f = 1 )</td>
</tr>
<tr>
<td>( Rv )</td>
<td>( v_0 = -9.040, b = 5.469, g = 0.605, \tau = 1.119, f = 1 )</td>
</tr>
<tr>
<td>( Sg )</td>
<td>( v_0 = 1.383, b = -51.030, g = 36.898, \tau = 7413.10, f = 1 )</td>
</tr>
<tr>
<td>( \theta )</td>
<td>( \theta = 0.4859 )</td>
</tr>
</tbody>
</table>

Figure 4.1: The structure of the best evolved CTRNN network performing the two-way distance measuring task: moving forward until the food sensor is activated then returning to the location it began from and signalling. Fo, food sensor; Sp, speed sensor; Fw, forward motor neuron; Rv, reverse motor neuron; Sg, signal neuron. \( w_{X,Y} \) indicates a weight from component X to component Y. For weight and and neuron parameter values see Table 4.1.
Figure 4.2: The dynamics of the best evolved agent for the two-way distance measuring task. Fw, output of the forward motor neuron (left axis); Rv, output of the reverse motor neuron (left axis); Sg, output of the signal neuron (left axis); A, position of the agent relative to the starting position (right axis). Horizontal lines mark the nest position (0, right axis) and the signalling threshold (0.4859, left axis). The agent is visiting and returning from a food item placed at 400 distances units (the maximum experienced during evolution). The agent signals when the signal neuron output crosses the signal threshold in an upward direction approximately at time 836. The agent crosses the nest location approximately at time 837.

plain the existence of the four remaining weights: those from the forward and signal neurons to the reverse neuron, and those from the signal and reverse neuron to the forward neuron. Simply removing these weights renders the agent non-functional, but since it is possible that compensatory changes to the remaining parameters could compensate for this, we cannot conclude with any certainty that they are truly required for the network to function. The functionality of these remaining weights is investigated using manual modifications to the evolved network, and the conclusion is reached that certain of the other weights fine tune the agent’s behaviour in such a way as to increase its fitness.

A plot was produced showing the agent’s error signalling the nest location as a function of the distance to the food, with wind speed set to zero (i.e. the average wind speed experienced during evolution). The food distance was sampled every 5 distance points from 5 to 400. The signalling error was defined as the location of the agent when it signalled, zero being the correct location. A series of modifications were made to the network, and the error graph replotted to compare performance to the unmodified network. Fig. 4.3 shows the signalling error of the agent for the original network and five modified versions.

The reverse motor neuron is to a very good approximation turned off when the food sensor is off and turned on when the food sensor is on. This means it cannot be playing an active role in the dynamics, apart from when it transitions, since any minor changes in its input will not be reflected in its output. This suggested removing all incoming weights to the neuron except those from the food sensor. In addition the weight from the food sensor was set to 100, and the neuron’s gain and
Figure 4.3: Plot of the error in the agent’s signalling location for the original evolved network and five hand-modified versions of the network, as a function of the distance from the starting location to the food. The correct location to signal is zero. A, the unmodified evolved network; B, reverse neuron modified; C, forward neuron modified; D, forward and reverse neuron modified; E, rescue I; F, rescue II. See text for details of modifications applied.
bias set to 1 and $-50$ respectively (to ensure the neuron was fully saturated). This change altered
the error curve only slightly relative to the effect of the other modifications (Fig. 4.3, B), making
the agent less accurate in the range $[0, 65]$ distances units, more accurate in $[65, 135]$, less accurate
in $[135, 340]$ and more accurate in $[340, 400]$, resulting on average in a small reduction in accuracy.
To a good approximation the incoming reverse neuron weights, except that from the food sensor,
can therefore be considered redundant features, and it can be considered to simply map the output
of the food sensor to the reverse thruster, and to provide part of the integrator’s offset.

Next the same change was made to the forward motor neuron, which was to a first approxima-
tion ‘on’ (outputting 1.0) during the first leg and ‘off’ (outputting 0.0) during the second leg of the
journey. However, for journeys longer than about 50 units, the output began to decrease slightly
towards the end of the first leg. For journeys of 400 units (the maximum) the output had fallen
from 0.998 to level off at around 0.97 by the end of the first leg. The shape of this decrease was
noticeably similar to that of the output of the signal neuron. Since the time constant of the forward
neuron is very low, and since it has constant inputs from the food sensor and reverse neuron, it
can only be responding reactively (i.e. in a non-integrative way) to changes in the output of the
signal neuron. This explains why the decrease levels off because, although the internal state of the
signal neuron keeps decreasing as long as the agent moves forward, the output is sigmoidal and
levels off as the agent reaches 300 distance units. The forward neuron’s incoming weights were
zeroed except that from the food sensor, which was set to $-100$, its gain and bias were set to 1 and
50 respectively (again, making the neuron saturate on or off under the control of the food sensor).
The error curve was noticeably different (Fig. 4.3, C), having a steadily increasing gradient, upto
a maximum error of around 22 distance units for the longest journey (11.7 times the error of the
unmodified network at that distance). The error was significantly greater than the unmodified net-
work for all distances. The curve was very similar whether or not the modification to the reverse
neuron was applied at the same time (Fig. 4.3, D) or reverted back to the unmodified state (Fig.
4.3, C). Thus we suspect that the forward neuron’s function is slightly more complex than to map
the food sensor output to the forward thruster and provide part of the integrator neuron’s offset.

The next two modifications attempted to rescue the modified agent by compensating for the
two changes made so far. As explained above the signal neuron requires both an input from the
speed sensor and a roughly constant offsetting input, which it gets from the forward and reverse
neurons. This suggested setting the weights from both of those neurons to the signal neuron to the
same value (since we expect the offset to be constant whether provided by either neuron). First
the weight from the forward neuron was set to the value of the weight from the reverse neuron
(10.781). This resulted in a curve of a similar shape to the double modified network (both forward
and reverse neurons modified to saturate), but whose error was in the opposite direction (Fig. 4.3,
E): decreasing down to a value of -20 for a distance of 400 units. This suggested setting the value
of the forward-to-signal neuron weight to the average of its unmodified value and that from the
reverse neuron, $\frac{11.302 + 10.782}{2} = 11.042$. This resulted in a shallow U-shaped curve of much better
accuracy than the first rescue network or the modified network, but still significantly worse than
the evolved network (Fig. 4.3, F).

The combined input to the signal neuron from the forward and reverse neurons over time for
the (unmodified) network is not constant. For a maximum length journey, performed with wind
speed set to zero, the offset begins at around 11.3, drops in a sigmoidal shape to 11 at a distance of 300 (due to the influence of the signal neuron on the forward neuron), then drops suddenly (as the food sensor activates) to a value of 10.8 for the return journey. The value begins to increase in a sigmoidal shape again at the very end of the journey, but the agent reaches home before this increase becomes significant. The ‘expected’ value for the offset, half the value of the weight from the speed sensor, is 9.66. The value is therefore consistently high, and more so during the start of the outward leg.

This can be seen as a way to counteract the effects of the decay of the integrator neuron state. The state of this neuron begins at 1.383 and decreases to a minimum of about 1.130 when the agent is at a food item 400 distance units from the nest. The CTRNN decay term (see Eqn. 3.1) therefore acts to decrease the value of the neuron’s state during both legs of the journey. The effect is strongest when the integrator state is highest, at the beginning and end of the journey. Shifting the offset to a larger magnitude than 9.66 acts to increase input to the integrator and thus counteracts the decay effect. The larger initial shift appears to compensate for the greater initial decay effect due to the larger value of the integrator’s state at the journey’s start. The initial value of the integrator neuron is slightly higher than the threshold value, and this could also be a mechanism to counteract the effects of decay since the integrator state then does not need to return all the way to its initial value in order to signal. However the difference between the initial state and the value required to signal is not significant relative to the change in integrator state which occurs for anything but the shortest journeys. Hence the difference between 9.66 and the actual ‘offset’ value can be considered as the most important mechanism to counter the decay of the integrator.

The excess offset (the combined input to the signal neuron from the forward and reverse neurons, minus negative one half the value of the weight from the speed sensor) required to exactly counteract the decay term is simply equal to the negative of the signal neuron state, provided we assume that the change in integrator state must be proportional to the agent’s location. The state equation for the signal neuron is as follows:

$$7413 \dot{v}_{Sg} = -v_{Sg} + w_{Sp,Sg}p + w_{Fw,Sp}Fw + w_{Rv,Sg}Rv$$

We can easily see that for the right handside to approximate $w_{Sp,Sg}(Sp - \frac{1}{2})$ the value of $w_{Fw,Sp}Fw + w_{Rv,Sg}Rv$ must be equal to $v_{Sg} - \frac{w_{Sp,Sp}}{2}$, where $-\frac{w_{Sp,Sp}}{2}$ is the ‘expected’ offset value and $v_{Sg}$ is the ‘excess’ required to counteract the signal neuron decay term. Fig. 4.4 shows a plot of the integrator state along side the excess offset. The two roughly coincide as expected.

Since the gradual changes in output of the forward neuron are caused by input from the signal neuron, and the changes feed back into the signal neuron, the mechanism is very similar to a self weight on the signal neuron. Such a self weight is found in the results of the one-way task experiments.

### 4.3.2 The One-way Task

For the one-way task, the best of the five evolved agents was examined. Fig. 4.5 shows the network, Table 4.2 shows the parameter values. A simple analysis follows explaining how the network functions. Following this, once again a series of alterations are made to the network to probe its
Figure 4.4: Plot of the signal neuron state (thin line) and the excess offset value (see text) being inputted to the signal neuron (thick line) over the course of a journey to and back from a food item placed at 400 distance units from the nest. The wind speed was set to zero. Were the network to exactly compensate for the CTRNN decay term, the two values would be identical over the entire journey.
function in detail to determine which weights are redundant and which increase the agent’s fitness in ways neglected in the initial simple analysis.

The speed sensor was connected to the forward motor neuron in such a way as to implement a negative feedback loop, causing the agent to travel at an approximately constant speed in the face of the wind force. The signal neuron decayed at an approximately linear rate from its initial state while the food sensor was inactive, but acted to integrate the food sensor output once it was active at an equal but opposite rate. This approximates the integration of the time elapsed during each journey leg such that the signal will be produced at the correct location provided both legs took an equal time to complete. The negative feedback system acting on the agent’s speed ensures that this condition is approximately true regardless of wind.

A plot of the error in the position signalled by the evolved agent with wind set to zero (the average wind speed experienced during evolution) throughout the journey shows a smooth curve of values ranging from -2 to 2.5 (Fig. 4.7, lower thin line). The output of the reverse motor neuron is zero throughout all trials investigated, and upon removing this neuron and all its incoming and outgoing weights the signalling accuracy plot was exactly as for the evolved network. This simplifies the network structure since we now only need to consider four weights: from the speed sensor to the forward neuron, from the food sensor to the signal neuron, from the signal neuron to itself and from the signal neuron to the forward neuron.

Removing the signal to forward neuron weight resulted in only a slight change in the graph (Fig. 4.7, lower thick line). This weight is of a much lower magnitude (0.647) than that from the speed sensor (-19.497). In the presence of wind the output of the forward neuron spends most of its time saturated at maximum or minimum (Fig. 4.6, thin line), and switches rapidly between the two in response to changes in speed sensor output. With wind speed fixed at zero, after an initial period showing damped oscillations (not shown) the output settles down to a curve...
Figure 4.6: Internal dynamics of the CTRNN network performing the one-way task. The agent is visiting a food item 400 distance units from the nest, then continuing for a further 400 units. Thin jumpy line, output of forward motor neuron (left axis); thick diagonal line, position of agent (right axis); thick curved line, output of signal neuron (left axis); upper horizontal line, correct signalling location at 800 distance units (right axis); lower horizontal line, signal neuron threshold (left axis).

<table>
<thead>
<tr>
<th>Component</th>
<th>Parameter Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>(w_{Sp,Fw})</td>
<td>(w = -19.497)</td>
</tr>
<tr>
<td>(w_{Sp,Rv})</td>
<td>(w = 3.740)</td>
</tr>
<tr>
<td>(w_{Fo,Rv})</td>
<td>(w = -0.740)</td>
</tr>
<tr>
<td>(w_{Fo,Sg})</td>
<td>(w = 13.602)</td>
</tr>
<tr>
<td>(w_{Fw,Rv})</td>
<td>(w = 19.850)</td>
</tr>
<tr>
<td>(w_{Rv,Fw})</td>
<td>(w = -6.836)</td>
</tr>
<tr>
<td>(w_{Sg,Fw})</td>
<td>(w = 0.647)</td>
</tr>
<tr>
<td>(w_{Sg,Rv})</td>
<td>(w = -24.620)</td>
</tr>
<tr>
<td>(w_{Sg,Sg})</td>
<td>(w = 0.601)</td>
</tr>
<tr>
<td>(\dot{F_{w}})</td>
<td>(v_0 = -12.84, b = 1029.60, g = 80.168, \tau = 57.810, f = 1)</td>
</tr>
<tr>
<td>(\dot{R_{v}})</td>
<td>(v_0 = -154.344, b = 14404.531, g = -0.947, \tau = 839.46, f = 0)</td>
</tr>
<tr>
<td>(\dot{S_{g}})</td>
<td>(v_0 = 7.10, b = -84.01, g = 11.830, \tau = 8912.51, f = 1)</td>
</tr>
<tr>
<td>(\theta)</td>
<td>(\theta = 0.4942)</td>
</tr>
</tbody>
</table>

Table 4.2: Parameter values of the evolved CTRNN network performing the one-way distance measuring task. Fo, food sensor; Sp, speed sensor; Fw, forward motor neuron; Rv, reverse motor neuron; Sg, signal neuron. \(w_{X,Y}\) indicates a weight from component X to component Y. \(\theta\) is the signalling threshold of the signal neuron. All neurons with the flag \(f\) set to 1 are centre-crossing and for these the values shown for \(v_0\) and \(b\) are the centre-crossing ones. See Fig. 4.5.
Figure 4.7: The location of the agent when it signalled as a function of the distance from the nest to the food item. Zero (left axis) is the correct place to signal. The wind speed was set to zero for all journeys. Lower thin line, the evolved network; lower thick line, the network with weight $w_{Sg,Fw}$ removed; upper thin line, the network with weight $w_{Sg,Sg}$ removed; upper thick line, the network with both $w_{Sg,Fw}$ and $w_{Sg,Sg}$ removed. In all cases the behaviour of the network was unchanged if the reverse neuron and all its incoming and outgoing weights were removed.

which resembles the output of the signal neuron, rescaled to the approximate range $[0.64, 0.7]$. This suggests the agent will travel slightly faster at the beginning and end of journeys, with this feature being more pronounced for longer journeys. It is not clear if this increases the accuracy of signalling significantly.

Removing just the signal neuron’s self weight, the error plot was changed considerably (Fig. 4.7, upper thin line). The error now ranged from almost zero for short journeys up to eleven for journeys of 250 distance units, and overall was much worse than the evolved network. The state of the signal neuron ranges from an initial value of 7.1 down to a minimum of 6.6 for the longest journeys. The initial value and time constant (8912.51) mean that on the outward journey decay is roughly linear. The weight from the food sensor is 13.6 which is very close to twice the midpoint of the signal neuron state’s range $7.1 + 6.6 = 13.7$. The effect of food sensor activation is therefore to negate the decay term ($-v \in [-7.1, -6.6]$) by adding a value twice as large but opposite in sign, and therefore making the neuron state change during the second journey leg at the same rate but in the opposite direction as during the first leg of the journey. The effect of the small self weight is to compensate for the change in magnitude of the decay term, which ranges from $-7.1$ to $-6.6$. Since signal neuron output is largest when the decay term is largest an appropriate positive self weight will roughly counteract the change. Signal neuron output ranges from 0.5 to 0.0. 0.5 times the self weight gives a contribution of 0.3 which is of the same magnitude as the changes in the decay term $7.1 - 6.6 = 0.5$.

A 3D plot of the agent’s speed, the forward motor output and the wind speed against each other is shown in Fig. 4.8. Beginning at the middle left of this plot, where wind speed is low, motor output is high and speed is low. The high motor output is acting to increase the speed of the
agent towards a medium value. Some of the trajectories jump up due to large increases in wind speed. These then move to the right along the top of the diagram as the increased wind increases the agent’s speed. If wind speed remains high these move down and right, indicating that motor output is dropping in response to the high agent speed, causing a decrease in speed. From the middle right of the diagram we can follow trajectories around the other half of the diagram in a manner analogous to that just described, in response to a large decrease in wind speed.

At low wind speeds and low agent speeds we see motor output increasing quickly, at high wind speeds and high agent speeds we see motor output decreasing quickly. For a narrow band of intermediate agent and wind speeds (centred roughly around an agent speed of 0.625) we see the motor neuron at intermediate values.

The negative feedback system acts to regulate the agent’s speed throughout the journey. This comes at the cost of not travelling as fast as possible, and therefore of taking longer to complete the task (and getting a slightly lower fitness). The simplest system of this kind might have the agent travel at its fastest guaranteeable speed. Assuming the agent’s maximum speed is \( A \) in either direction and that of the wind is \( B \) in either direction, the maximum speed the agent can guarantee travelling at is \( A - B \) in either direction. Since the wind’s maximum speed is in fact equal to that of the agent we cannot guarantee a non zero speed with absolute certainty, but could expect to be able to travel at a desired speed \( d \) at a random point during a journey with probability \( 1 - \frac{|d|}{2} \). The evolved agent was measured as having taken 800 time units to travel a distance of 500 with wind as during evolution (data not shown), and therefore travels at an average speed of approximately 0.625, giving a probability of 0.6875. This means that when wind speed is less than -0.375 the agent must travel more slowly than its apparent average speed. The evolved agent does better than simply suffering this reduction in speed passively. A plot of distance over time (Fig. 4.6) shows that the agent does indeed sometimes travel at less than the average speed, but that after this period has ended it spends a time travelling at above average speed to compensate once wind speed has

Figure 4.8: 3D plot of the agent’s speed, the forward motor output and the wind speed during a journey of total length 800 distance units. See text for discussion.
become permissive. The forward motor neuron time constant is 57.81. A CTRNN time constant defines the time the state takes to move $1 - e^{-1} = 63.2\%$ of the way towards its new “equilibrium” value when its input changes (assuming it was at equilibrium before the change occurred). Since the wind changes at least every 20 time units, it is clear that the neuron will not generally come to equilibrium, and that therefore its value will continue to change during each wind speed epoch. The consequence is that the longer the wind speed has been at an extreme value, the longer the forward neuron will take to respond once it changes to a less extreme value. Therefore the negative feedback effect induced by a large negative wind speed (the forward neuron will saturate on) will persist after it has ceased, causing the agent to travel at above average speed for a spell, the duration of which will be positively correlated with the duration of the large negative wind speed epoch.

4.4 Conclusions

The GA was able to evolve CTRNN networks capable of solving both tasks using only three neurons. The two-way task was solved most efficiently, since the agent could travel at its maximum available speed at all times, but still signal the goal location accurately. The average speed of the one-way task agent was limited to around 60 per cent of the possible maximum, since this allowed it to travel at a more uniform speed as required by its signalling strategy.

The two-way agent integrated the output of the speed sensor plus an offset value to give the signal neuron state a value that was approximately a linear mapping of the agent’s spatial location. Accuracy was improved by varying the offset value to compensate for the leakiness of the integrator, and by making the initial value of the integrator state slightly larger than the value needed to trigger signalling.

The one-way agent essentially used two uncoupled (when considering only the controller) mechanisms. The first caused the agent’s average speed to be 0.625 (where 1.0 is the maximum average speed expected given an average wind speed of zero). This was achieved partly by regulating the speed towards this value, and partly by actively exceeding this speed to compensate for periods when the speed was forced below it due to strong opposing wind. The second mechanism counted the time elapsed during the first leg, and then counted in the opposite direction during leg two, signalling when the counter reached the threshold value. A self weight on the integrating signal neuron reduced errors due to integrator leakiness.
Chapter 5

Two Dimensional Path Integration. I

5.1 Overview

This chapter details the main sequence of experiments for this thesis. The task for the simulated agent is to perform PI-mediated homing in a two dimensional arena after an initial random excursion. This task is differentiated from the task presented in the next chapter by the type of sensors available. This chapter the agent has compass sensors, returning values related to the agent’s current heading relative to the fixed coordinate system of the external environment. The next chapter the agent has rotation rate sensors, and no direct access to information about its absolute compass heading. In both cases the agent also has a linear speed sensor and two beacon (light) sensors.

Given five sensors (two compass sensors, two beacon sensors, one speed sensor), and three output neurons (two opposing rotation rate controllers and one controlling the forward speed) GeNGA is used to evolve a neural controller capable of generating homing behaviour for a variety of experimental setups. CTRNN and ModCTRNN are compared as control systems. The exact form of the compass sensor response function is varied. The agent is either allowed to travel at a constant speed throughout its journey (thus simplifying the process required to update the HV) or is forced to travel at varying speeds by the application of motor noise and holding the agent still for a random time at one point during the journey.

GeNGA only succeeds into producing a CTRNN controller when the agent’s speed is constant, and even here fails to make the agent home in an efficient straight line to the nest. The controller is analysed. ModCTRNN evolves a highly successful controller, which works whether or not the agent’s speed varies, and causes efficient straight homing. Upon reaching the vicinity of the nest, the controller produces a search-like behaviour. This is achieved when the compass response function is sinusoidal. The performance of GeNGA at evolving solutions degrades when alternative compass response functions are used.

The successful ModCTRNN controller is analysed in detail and found to implement an augmented version of the Mittelstaedt geocentric Cartesian PI model. The model is extended in two ways. Firstly, a mechanism performs “leakage correction” and allows accurate navigation to take place despite the HV being stored on leaky (decaying) integrators. Hence, here the presence of leaky integration does not introduce systematic homing errors. Secondly, the output of the Mit-
telstaeedt system is filtered through the rotation control neurons in such a way as to modify its spirograph-like trajectories into a more efficient searching behaviour.

5.2 Methods

5.2.1 GeNGA

GeNGA with a population of 30 genotypes was used, each encoding a neural network. Each genotype was evaluated in 10 independent trials per generation. Each trial, the agent, controlled by the encoded network, was required to perform the PI task, and was assigned a fitness value. The genotype’s overall fitness was the mean fitness over the 10 trials. The fittest five genotypes were retained unmodified in the population each generation. Each was also copied five times to produce 25 new genotypes, which were mutated and used to replace the 25 least fit genotypes. As well as mutating the parameter values of the network, GeNGA was also used to change the number of neurons and weights, present in the network (see Section 3.1).

5.2.2 The Path Integration Task

All differential equations were numerically integrated using the Euler first order method with a step size of 0.001 during evolution, and successfully evolved agents were checked for numerical integration artefacts using a step size of 0.0001.

For each trial, the agent started at the nest with a random orientation and was presented with a series of between one and three visual beacons that it was required to visit (defined as approaching to a distance of 0.01 or less) by phototaxis. Each beacon was immediately removed when the agent reached it, and the next one activated. Beacons were placed by selecting a random distance from the nest in the range \([0.5, 1.0]\) and random angle from the nest in the range \([0, 2\pi]\) radians, except the last beacon’s angle, which was selected using a stratified random scheme that divided \([0, 2\pi]\) into 10 equal-sized blocks, one for each trial, thus ensuring a more even coverage of final HVs per evaluation. After the last beacon was removed, the agent’s orientation was randomised (to prevent it homing by simply turning through \(\pi\) radians for a one-beacon trial) and it was required to return to the nest (again defined as approaching within 0.01 distance units) using only its compass sensors, speed sensor and internal state, i.e. the nest could not be directly detected by the agent.

During a given generation, all 30 agents were presented with the same 10 sets of beacon locations and given the same initial orientations at the nest and were subjected to identical noise. This made fitness comparisons less subject to noise. Each experiment was initially performed with the agent’s speed fixed at maximum, so that a solution could evolve without requiring the speed sensor. If the GA produced a satisfactory solution, the experiment was repeated with the agent’s speed no longer restricted to maximum. The agent was now also held captive (stationary) for a randomly selected time drawn from \([0, 0.5]\) and at a randomly selected orientation drawn uniformly from \([0, 2\pi]\) upon reaching the last beacon before being allowed to attempt homing. If this task was solved, the experiment was continued with an increase in the level of noise on the motor outputs (see below), so forcing the agent to take full account of the speed sensor in order to perform accurate PI.
5.2.3 Calculating Fitness

The fitness assigned for each trial is calculated by one of four equations depending on how many phases of the task were completed by the agent. The overall fitness is always between zero and one, and the fitness for reaching each new phase is always greater than or equal to achieving optimal fitness in all previous phases. Firstly, if the agent failed to reach the first beacon within twice the time it would have taken to visit all of the trial’s beacons by a direct course at maximum speed the trial was ended and the fitness was:

\[ 0.25/(1 + \varepsilon_B) \]

where \( \varepsilon_B \) is the time integral of the agent’s Euclidean distance from the first beacon over the trial (a measure of the average distance from the beacon). Secondly, if the agent reached the first beacon but failed to reach all subsequent beacons within the above time limit, the trial was ended and the fitness was:

\[ 0.25 + 0.25 \times \frac{\text{visited}}{\text{total}} \]

These two criteria simply facilitate the evolution of phototaxis. Thirdly, if the agent visited all beacons but failed to reach the nest within three times the minimum time required from the location of the last beacon at full speed, the trial ended and the fitness was:

\[ 0.5 + 0.25/(1 + \varepsilon_N) \]

where \( \varepsilon_N \) is the integral of the Euclidean distance of the agent from the nest starting from the moment it arrived at the last beacon until the end of the trial. If the agent returned to the nest, its fitness was:

\[ 0.75 + 0.25/(1 + t_N) \]

where \( t_N \) is the time taken to complete the entire trial; thus, the fittest possible agent would visit all beacons and then return to the nest using straight, direct paths for all legs of the journey and travel at full speed. Due to sensor noise, cumulative navigation errors must arise, meaning that an ideal agent would also search efficiently for the nest after reaching the HV zero point until it found the nest or until the trial timed out.

5.2.4 The Agent

The agent was modelled as having a position \((x_A, y_A)\) on an unbounded two-dimensional plane, with orientation \(\theta_A\) radians measured positive anticlockwise from the x-axis (or “east”; all angles are defined positive anticlockwise from the x-axis or from the agent’s body axis where appropriate). The origin \((0, 0)\) of the plane was defined as the nest location. One distance unit was defined as the maximum distance the agent was ever required to travel from the nest. Maximum speed was also unity; the agent therefore took at least one time unit to reach maximum distance from the nest. The agent had two beacon sensors (for phototaxis), two (sometimes more) compass sensors, one speed sensor, one food sensor, two rotation motors and one forward motor. The agent’s motion was restricted to rotation and forward translation. No backward or sideways movement was allowed, in keeping with the usual behaviour of foraging ants. The agent was assumed to have low inertia; motor output therefore specified forward speed and rotation directly, rather than supplying forces. Speed was controlled by the forward motor neuron firing rate, \(F\), and rotation by the two opposing left and right rotation motor neuron firing rates, \(R_L\) and \(R_R\):

\[
\frac{d\theta_A}{dt} = 150(R_L - R_R)
\]
giving a maximum rate of turn of 150 radians per time unit, and maximum speed of 1 distance unit per time unit. This scheme was chosen, rather than simply simulating a two wheeled robot, so that the agent’s speed could be easily fixed at maximum without preventing it from steering. The agent’s body, consisting of sensors, motors, neurons and weights, was constrained to bilateral symmetry. Aside from three single-copy components (the speed and food sensors and the forward motor neuron) all components were created as symmetrical pairs, including all other neurons, sensors and weights. Compass sensor neurons had an activation function, \( f \), which responded maximally to a particular agent orientation, \( \theta_A = a \), and declined in a symmetrical way either side of this value. Each pair of compass sensor neurons had complementary values such that if one responded maximally at \( \theta_A = a \), the other responded maximal to \( \theta_A = -a \). This could be considered as the response of light sensors to a distant light in the east. All activation functions define the stimulus-response properties of the sensory neuron in terms of its firing rate. The function \( f \) was varied between experiments and was mostly based on the cosine function (see Fig. 5.1). As can be seen, some functions have negative as well as positive firing rates. This unbiological feature can be removed by replacing each compass sensor with two sensors, giving complementary half-wave rectified output. Negative values were used to simplify the analysis of the evolved networks. The agent was either given one pair of compass sensors with maximum responses set to \( \theta_A = \pm \pi/4 \) or had an evolvable number of sensor pairs with evolvable maximum response parameters. The pair of beacon sensor neurons was defined as having activation \( [\cos(\theta_B - \pi/2)/2] + 0.5 \) for the left sensor and \( [\cos(\theta_B + \pi/2)/2] + 0.5 \) for the right, where \( \theta_B \) is the angle to the current beacon from the agent’s body axis (the activation is therefore independent of distance to the beacon). Additionally a food sensor was used: its activation was 0 before reaching the final beacon (where the agent might be imagined to have collected an item of food to motivate its return to the nest) and 1 thereafter (this sensor was ignored by most of the evolved networks). The speed sensor neuron gave a proprioceptive measure of the agent’s current speed, including motor noise, linearly mapped to the range \([0, 1]\). Noise was applied to all initial neuron states \((v_{t=0})\), sensors and motors by adding a uniformly distributed random offset in the range \([-\eta, \eta]\). The offsets for sensors and motors where held constant across multiple numerical integration steps; changes in value were triggered using a Poisson process rate \( r \) (the average number of events per unit time) to calculate the time of the next value change: \( t_{\text{change}} = [-\log(1-x)/r] + t \), where \( x \) is a uniformly distributed random value drawn from the interval \([0, 1]\), and \( t \) is the current time. The values of \( \eta \) and \( r \) could be set for each sensor/motor type; \( \eta = 0.01 \) and \( r = 20.0 \) were used initially for all sensors and motors; for some experiments, this was increased part way through for the forward motor neuron to \( \eta = 0.7 \), \( r = 2.0 \) and for the rotation motor neurons to \( \eta = 0.1 \), \( r = 20.0 \) in order to impose large long lasting variations in the agent’s velocity. Overall, the presence of noise in the simulation is intended to promote the evolution of robust PI solutions and to cause small cumulative navigation errors, as occur in natural PI.

The use of Poisson process events to trigger discontinuous changes in the noise offset values may seem unnatural in a continuous time simulation. A more conventional approach would be to use an Ornstein-Uhlenbeck process for each noise value. The advantage of using the present approach is that the behaviour of the system converges onto a single behavioural trajectory as the step size is decreased, provided only that the random number generator (the Mersenne twister
Figure 5.1: The four compass response functions used. From the bottom: cosine, positive cosine, piecewise linear approximation of cosine, head direction cell. Plot of sensor output (y-axis) against $\theta_A - a$ (x-axis), where $\theta_A$ is the agent’s current orientation and $a$ is the sensor’s preferred direction.

Matsumoto and Nishimura, 1998, is used here) is seeded with the same value. This is simple to achieve using the Poisson approach since the random number generator is only called once for each noise event, whereas Ornstein-Uhlenbeck uses a random value each integration step and hence it calls the generator more and more frequently as the step size decreases. It converges to the same behaviour statistically as step size decreases, but never to a single trajectory.

The neural network used to control the agent was constrained to bilateral symmetry by means of generating a symmetrical pair of neurons for every neuron encoded in the genotype. This feature seems to have been important in the ability of the GA to solve the task since earlier attempts (not detailed here) not using bilaterally symmetrical control networks failed to produce significant results.

5.2.5 Experiment 1A
The agent was given one pair of compass sensors giving a cosine-shaped response (see Fig. 5.1). The CTRNN model was used. The agent always moved at its maximum speed. Once the agent had evolved to solve the task reliably, the GA was run in pruning mode until the size of the genotypes stabilised, indicating that most or all network redundancies had been removed.

5.2.6 Experiment 1B
As experiment 1A except that the agent’s speed was not fixed at maximum and the agent was held stationary for a random interval at the final beacon.

5.2.7 Experiment 2A
As experiment 1A except that the ModCTRNN model was used.

5.2.8 Experiment 2B
As experiment 2A except that the agent’s speed was not constant and the agent was held captive for a time at the last beacon. Once the agent had evolved a good fitness, a further perturbation was
introduced by setting $\eta = 0.7$, $r = 2.0$ for the forward motor neuron (forcing large long-lasting changes to the agent’s speed via the noise offset mechanism) and $\eta = 0.1$ for the rotation motor neurons. Most of the Results section is dedicated to an in-depth analysis of the network evolved in this experiment.

5.2.9 Experiment 3A
As experiment 2A except that compass sensor responses were of the linear cosine type (see Fig. 5.1) and the number and maximum response direction of the compasses were allowed to evolve (up to a maximum of 10 compass pairs).

5.2.10 Experiment 3B
As experiment 3A except that the agent’s speed was variable, as in experiment 1B.

5.2.11 Experiment 4
As experiment 3A except that the compass sensor response was the “head direction cell” type (see Fig. 5.1), intended to model the response properties of head direction cells in rats Taube (1997).

5.2.12 Experiment 5
As experiment 3A except that the compass sensor response was the positive cosine type (see Fig. 5.1).

Three replicates were performed for each experiment. Results are described for the most successful of the three runs.

5.3 Results
See Table 5.3 for a summary of the results.

5.3.1 Experiment 1A: CTRNN constant-speed PI
The fittest agent after approximately 67000 generations returned to the nest within the time limit in 989 of 1000 test trials. The bilaterally symmetrical network (see Fig. 5.2) contained $12 \times 2$ weights (of which two pairs share the same source and target and so are not visible in the figure) and $3 \times 2$ interneurons. The shape of the return journey was highly dependent on the bearing to the nest from the last beacon and was generally not straight or direct (see Fig. 5.3 for one example) but usually consisted of two or more phases characterised by different patterns of oscillation in four of the neurons (shown in grey in Fig. 5.2; Fig. 5.4 shows the neural dynamics), resulting in various looping and zigzagging behaviours. The nonoscillatory neurons act as integrators of the compass sensor inputs. Their output and the current compass sensor output feeds into the oscillatory group. A plot of fitness against the angle to the beacon from the nest for 360 single beacon trials (data not shown) showed a clear sinusoidal-shaped relationship, caused by the agent taking longer to return to the nest from some regions than others. This was clearly a suboptimal solution that did not home as $C. fortis$ does in a straight line.
Table 5.1: Summary of results: Net.(network model used): C (CTRNN), M (ModCTRNN); Spd. (speed during trial): const. (constant), var. (variable); Comp.(compass response function, see Fig. 5.1): cos (cosine); lin. cos (linear approximation of cosine), HD cell (head direction cells), pos. cos (positive cosine); Returns (percentage of successful returns to nest out of 1000 test trials); Size (number of interneurons and weights). Results shown are for the best network obtained from three replicate experiments.

<table>
<thead>
<tr>
<th>Exp</th>
<th>Net</th>
<th>Spd</th>
<th>Comp</th>
<th>Returns</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1A</td>
<td>C</td>
<td>const.</td>
<td>cos</td>
<td>98.9%</td>
<td>6.24</td>
</tr>
<tr>
<td>1B</td>
<td>C</td>
<td>var.</td>
<td>cos</td>
<td>0%</td>
<td>-</td>
</tr>
<tr>
<td>2A</td>
<td>M</td>
<td>const.</td>
<td>cos</td>
<td>90.4%</td>
<td>0.8</td>
</tr>
<tr>
<td>2B</td>
<td>M</td>
<td>var.</td>
<td>cos</td>
<td>99.2%</td>
<td>0.12</td>
</tr>
<tr>
<td>3A</td>
<td>M</td>
<td>const.</td>
<td>lin. cos</td>
<td>39.9%</td>
<td>0.6</td>
</tr>
<tr>
<td>3B</td>
<td>M</td>
<td>var.</td>
<td>lin. cos</td>
<td>8.2%</td>
<td>0.8</td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>const.</td>
<td>HD cell</td>
<td>5.6%</td>
<td>0.12</td>
</tr>
<tr>
<td>5</td>
<td>M</td>
<td>const.</td>
<td>pos. cos</td>
<td>0%</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 5.2: Experiment 1A CTRNN network solving the constant speed PI task. $B_{L/R}$, left/right beacon sensor; $C_{L/R}$, left/right compass sensor; $R_{L/R}$, left/right rotation motor neuron; $N_{L/R}$, interneurons. Arrows are directional weighted links; double headed arrows indicate two links running in opposite directions. Grey circles are neurons involved in the oscillations responsible for generating the various modes of behaviour seen in this agent during its return journey (see Fig. 5.3).
Chapter 5. Two Dimensional Path Integration. I

Figure 5.3: Experiment 1A CTRNN agent performing the constant speed path integration task. $B_1$ indicates the single beacon. Beacon and nest are drawn as circles radius 0.01.

Figure 5.4: Experiment 1A CTRNN network dynamics for oscillating neurons $N_{R/L_3}$ and $R_{R/L}$ during return phase of journey (see Fig. 5.3). Plots are of neuron firing rates, the y axis runs from 0 (bottom) to 1 for each. $B_1$ and nest indicate the time of arrival at the beacon and the nest respectively (the nest is reached at the very end of the trial). $N_{L3/R3}, R_{L/R}$ are as in Fig. 5.2.
Chapter 5. Two Dimensional Path Integration. I 106

Figure 5.5: Final average homing direction (see text) plotted as a function of the location of the beacon visited before homing commenced. The agent started from the nest \((0, 0)\) with a random orientation \(\in [-\pi, \pi]\). Beacons locations were sampled with a spatial frequency of 0.1 distance units. Each location was tested with 12 agent runs starting from different initial orientations.

It appeared that the oscillatory neurons caused the agent to travel in a certain direction by means of entering a certain pattern of oscillations of the rotation control neurons in feedback with the compass sensors. The pattern of oscillation obtained depended on the state of the integrator neurons (which output to the oscillatory group). The relative phases and proportion of activation of the rotation control neurons during the oscillations corresponded to the shape of the agents trajectory over one period of the cycle. Since the value of the integrator neurons changes much more slowly than that of the oscillatory ones, the behaviour was investigated by freezing the state of the integrator neurons and observing the oscillatory pattern which persisted after any initial transient had died away. This investigation was carried out using the Runge-Kutta fourth order numerical integration method with a step size of 0.001 and with all sensor and actuator noise disabled.

To map the agent’s homing behaviour in this manner, the agent was first allowed to approach a single beacon (radius 0.001) placed in the arena, starting from a random initial orientation. Upon reaching the beacon its integrator neurons were frozen at their current values and the agent was allowed to begin homing. The agent tended to settle into a repeating pattern of motion driven by the oscillatory group. The average direction of motion of the trajectory was recorded after allowing 3.0 time units for any transient behaviour to die away. An arrow was plotted at the location of the beacon just visited indicating the homing direction finally adopted by the agent. A grid of twenty by twenty arrows was created in this manner, covering the region \(x \in [-1, 1]\) and \(y \in [-1, 1]\) at intervals of 0.1 (Fig. 5.5). To test the dependence of the homing direction on initial orientation of the agent, each beacon location was sampled 12 times from different initial orientations. In agreement with the observed lack of straight line homing, this plot deviates significantly from a field of arrows directed accurately at the origin.

The arrow plot is symmetrical about the line \(y = 0\). One feature of the plot is a zig-zag shaped
Chapter 5. Two Dimensional Path Integration. I

Figure 5.6: Final average homing direction (see text) plotted as a function of the location of the beacon visited before homing commenced. The agent started from the nest \((0,0)\) with a random orientation \(\in [-\pi, \pi]\). Beacons locations were samples with a spatial frequency of 0.1 distance units. Each location was tested with 26 agent runs starting from different initial orientations. In addition, the agent’s orientation and the states of the two oscillatory interneurons and the two rotation control neurons were randomised upon reaching the beacon.

boundary running from top to bottom marking a sharp change in arrow direction between two main regions of arrows in the left and right half of the plot. Within these two regions changes of arrow direction are mostly gradual, and most locations have only arrows pointing in a single direction. Along the x axis there are locations with arrows pointing in two or more directions, indicating a dependence on the agent’s initial orientation. Overall the field of arrows does give the impression of converging on the origin at \((0,0)\).

This plot maps the most meaningful feature (the long-term homing direction) of the five variable dynamical system consisting of the agent’s heading, and the states of the four oscillatory neurons. The map is of the effect of changing two parameters of this system, namely the output of the two integrator neurons which input to the two oscillating interneurons. Strictly speaking we have only mapped the attractor that the system finds when we have let the agent approach a beacon placed at a given location, and allowed it to home normally. It is possible that randomising the five states, or carefully selecting them might reveal further variability.

Producing the same plot, but this time randomising the five variables upon reaching the beacon (setting the heading \(\in [-\pi, \pi]\) and setting all neuron states \(\in [-50, 50]\)), and sampling each location 26 times with different randomisation values, we see now most locations can produce homing behaviour in more than one direction (Fig. 5.6). There are regions to the left and right of the origin where the direction seems to spread over a continuum, whereas above and below the origin are two regions where the direction can be one of two widely separated directions. A diagonal cross shaped region, separating the four variable regions, contains locations with only one homing direction.

Superimposing normal homing trajectories (where the integrator neurons have not been frozen
Figure 5.7: Fig. 5.5 with five homing trajectories superimposed. Each trajectory begins at the location of the beacon the agent has just visited and continues for 5 time units. The agent’s HV neurons are not frozen after reaching the beacon, nor are any of its other internal states randomised. The agent began each journey from the nest with a random orientation. No sensory or motor noise is present.

at the beacon) shows that homing roughly follows the arrow field, but that sometimes the agent can continue in its homing direction longer than the field predicts, taking some time to respond to the change of output of the integrative neurons (Fig. 5.7).

5.3.2 Experiment 1B: CTRNN variable-speed PI
This experiment failed to produce any successful agents after running three independent populations for 35000 generations. Seeding the GA with the fittest genotype from experiment 1A also failed for three populations after 65000 generations.

5.3.3 Experiment 2A: ModCTRNN constant-speed PI
Experiment 2A produced good results and, although evolved independently, produced a very similar network to the 2B experiment. The 2A results are not presented in any further detail here, since they do not add anything to the 2B results. The 2A experiment was also repeated using the same settings except the maximum time constant values (τ and α) allowed were 0.01, thus preventing neurons and weights from acting individually as integrators, but the GA failed to find any solutions.

5.3.4 Experiment 2B: ModCTRNN variable-speed PI
Since this experiment provided the most interesting and complete set of results, a detailed and extensive analysis is presented in this section. The results for the remaining experiments follow after. First the behaviour of the evolved agent is presented. Then comparative topology experiments are used to show that all parts of the network are contributing towards the fitness of the agent. Next an analytical approach to understanding the network is taken, showing that the agent performs PI and
homing following the simplest form of the Mittelstaedt bicomponent model, but that this model has been augmented to allow accurate navigation using decaying HV values and to allow a more efficient searching behaviour in the vicinity of the nest. A simplified five variable model of the full evolved network accounts well for the search patterns.

The fittest agent (Figs 5.8, 5.9, 5.10, Table 5.2) after about 35000 generations returned to the nest within the time limit in 992 of 1000 test trials. After automatic stochastic pruning the network contained six weight pairs and no interneurons. The return journeys were approximately straight (see Fig. 5.9) considering the level of motor noise. To test for artefacts resulting from numerical integration, 500 trials were performed using an integration step size of 0.001 (as during evolution) and 500 more with 0.0001; the agent reached the nest in 488 and 489 trials, respectively, a non-significant difference \( \chi^2 = 0, P > 0.1 \); chi-squared test). To test whether the ModCTRNN network was significantly better able to evolve solutions to this task than the CTRNN model, five more replicates of experiments 1B and five of 2B were performed for 60000 generations each, evolving the ability to return home from a single randomly placed beacon with full motor noise applied. None of the 1B runs produced an agent that returned to the nest in any of 1000 test trials, but for all 2B runs (some of which were stopped early when it became clear PI had evolved) the agent returned home in greater than 50% of 1000 test trials. Taking a random course straight from the beacon would result in between 1.6 and 3.2 returns to the nest per 1000, therefore zero is worse and 500 much better than this random strategy. Five failures for the CTRNN and five successes for the ModCTRNN is a statistically significant difference \( P < 0.01; \) Fisher’s exact test).

Search patterns
The agent displays a searching behaviour once it has reached the vicinity of the nest (see Patarnello and Carnevali (1989) for an example of previous simulation work dealing with efficient searching behaviour). If the simulation is run with all sources of noise removed, and the nest is removed so that the agent never ends its journey, the agent simply keeps moving around an area close to

Figure 5.8: Experiment 2B ModCTRNN network. \( B_{L/R}, \) left/right beacon sensor, \( C_{L/R}, \) left/right compass sensor, \( R_{L/R}, \) left/right rotation motor neuron, \( S, \) speed sensor and \( F, \) forward motor neuron. Arrows are weights, double headed arrows are two weights going in opposite directions between the same end points. Lines ending in small squares are weights which modify other weights. \( w_{\text{L1/R1}}, \ldots, w_{\text{L6/R6}}, \) are weights. See Table 2 for parameter values.
Table 5.2: Parameter values of the evolved network from experiment 2B. See Fig. 5.8.

<table>
<thead>
<tr>
<th>Component</th>
<th>Parameter Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_{L1}/R1$</td>
<td>$w = 12.0720$</td>
</tr>
<tr>
<td>$w_{L2}/R2$</td>
<td>$\alpha = 8.4355, \beta = 0.0001$</td>
</tr>
<tr>
<td>$w_{L3}/R3$</td>
<td>$\alpha = 0.0123, \beta = 2.0477$</td>
</tr>
<tr>
<td>$w_{L4}/R4$</td>
<td>$\alpha = 5.1753, \beta = -98.7613$</td>
</tr>
<tr>
<td>$w_{L5}/R5$</td>
<td>$w = 65.9304$</td>
</tr>
<tr>
<td>$w_{L6}/R6$</td>
<td>$w = -3.5159$</td>
</tr>
<tr>
<td>$F$</td>
<td>$\tau = 0.0489, v_0 = 38.2195, b = 42.8689$</td>
</tr>
<tr>
<td>$R_{L/R}$</td>
<td>$\tau = 0.0106, v_0 = -3.6629, b = 0.2994$</td>
</tr>
</tbody>
</table>

Figure 5.9: Experiment 2B Agent (see Fig. 5.8) performing the PI task. $B_{1,2,3}$ indicate the order the beacons were presented in, beacons and nest are drawn as circles of radius 0.01.
Figure 5.10: Experiment 2B Network performing the variable speed path integration task. Plot shows weights (y-axis, see Fig. 5.8) over the whole trial (x-axis, see Fig. 5.9). $B_{1,2,3}$ show the approximate time of arrival at each beacon (the nest is reached at the very end of the trial). From about 2 time units into the trial until 3 time units, a large change in the values of $w_{L3/R3}$ reflects noise applied to the forward motor output as detected by the speed sensor. The second abrupt change in these values reflects the period of enforced captivity of the agent at the last beacon before homing begins.

the nest (up to a range of about 0.2 units from it) in an irregular pattern which never repeats (Fig. 5.11, Fig. 5.12). Kim and Hallam (2000) have suggested that the homing mechanism of PI might be enough to also explain searching behaviour, and this indeed seems to be the case here. No special signal is given to the agent when it returns close to the nest, yet searching behaviour begins automatically. The trajectory of the 2B experiment agent is more complex than the expected spirograph-like behaviour of the Mittelstaedt model (Figs 2.5, 2.6, 2.7 and 2.8) and shows little dependence on the amount of noise present (Fig. 5.13). Since the fourth fitness criteria rewards shorter total journey times, we have been implicitly selecting for efficient search as well as direct, rapid initial homing. Since the agent reaches home in $\approx 99\%$ of trials, the third fitness criterion, penalising higher average distances from the nest during homing if the agent did not reach home before the end of the trial, will not have been significant during the final stages of the network’s evolution, allowing broad search patterns to evolve if necessary. To compare the experiment 2B agent’s search with that seen in $C.\ fortis$, the average search density of 50 trials returning from a beacon 0.75 distance units east was plotted (Fig. 5.14), showing an approximately symmetrical bell shape. $C.\ fortis$ (Wehner, 1992) and other $Cataglyphis$ species (Wehner and Srinivasan, 1981) show a similar search profile, which maybe close to optimal.

Using the Runge-Kutta 4th order numerical integration method with a step size of 0.0001, the behaviour of the evolved network was investigated by means of varying parameter values. This suggests that the parameters have been selected to generate an irregular, fine grained space filling, non-repeating, roughly radially symmetrical search pattern centred on the agent’s estimate of the
Chapter 5. Two Dimensional Path Integration. I

Figure 5.11: A 3D $(t,x,y)$ plot of the unmodified agent returning from a beacon placed at $(0,0.5)$. The simulation used the Runge-Kutta 4th order numerical integration method with a timestep size of 0.0001. There are no sources of noise. The simulation lasted 50 time units.

Figure 5.12: A $(x,y)$ plot of the same journey as Fig. 5.11, where the unmodified agent returns from a beacon placed at $(0,0.5)$. There are no sources of noise. The simulation lasted 50 time units.
Figure 5.13: The experiment 2B agent's behaviour if the nest is removed (a) with all sources of noise removed and (b) with 1 percent sensor noise, 10 percent rotation motor noise and 70 percent forward motor noise (as during evolution). A searching behaviour is clearly visible which does not rely on the presence of noise, and arises from the coupled dynamics of the agent’s network and motion.

Figure 5.14: Experiment 2B Search density of the agent when the nest has been removed, averaged over the final 35 per cent of 50 independent trials. The agent is returning from a single beacon placed at a distance of 0.75 to the right in all trials. The white dot shows the fictive nest position, the black bar shows the size of the nest (diameter 0.02).
nest location. Changes to the value of the weight connecting the two output neurons, and to their
time constant parameter (values which are likely to influence the search pattern, see the section
on analytical results below) cause deviations towards more regular and less space filling and less
symmetrical patterns. The behaviour of the agent is shown after returning from a beacon placed
at (0, 0.5), the simulation lasts 100 time units. Figs 5.15, 5.16, 5.17, 5.18 show the behaviour
when the contralateral weight connecting the output neurons (whose evolved value is -3.5) has a
value of -1.0, -2.5, -4.5 and -5.0 respectively. Figs 5.19, 5.20, 5.21 show the behaviour when the
rotation control neurons time constant (normally 0.01) is 0.005, 0.02 or 0.04 respectively.

**Homing Accuracy**

It has already been shown (Chapter 2) that the Mittelstaedt PI model can generate systematic
navigation errors closely resembling those seen in *C. fortis* (Müller and Wehner, 1988) when the
HV is stored in leaky integrators. This section shows that, despite the fact that the HV of the
evolved agent is stored on leaky integrator weights with a relatively small time constant relative to
the average journey time, no significant homing errors are present. Later analysis (Section 5.3.4)
shows how a leakage correction mechanism achieves this.

Homing was investigated as follows. Before homing the evolved agent was made to visit two
beacons, the first placed a distance, \(A\), directly above the nest, the other placed a distance \(B\) and at
an angle \(\alpha\) (measured clockwise from the direction of the y axis) from the first beacon. This caused
the agent to follow an L shaped journey, much as the ants performed in Müller and Wehner’s
experiments, such that a comparison should reveal whether similar navigational errors exist. The
homing direction of the agent was defined as the compass heading of the line connecting the
location of the second beacon to the centre of the agent’s search pattern (using Müller’s convention
of positive bearings clockwise from downwards). The search pattern was defined to start when the
agent was 0.02 distance units from the nest, and continued until a total of 10 time units had elapsed
since the beginning of the journey. The centre was the average location during search. There were
no sources of noise in the simulation. The agent’s initial orientation was towards the first beacon.
Figure 5.16: The behaviour of the agent over 100 time units, returning from a beacon at (0, 0.5), with the contralateral weight connecting the rotation control neurons changed from $-3.5$ to $-2.5$. The length of the scale bar is 0.05.
Figure 5.17: The behaviour of the agent over 100 time units, returning from a beacon at \((0, 0.5)\), with the contralateral weight connecting the rotation control neurons changed from \(-3.5\) to \(-4.5\).

Figure 5.18: The behaviour of the agent over 100 time units, returning from a beacon at \((0, 0.5)\), with the contralateral weight connecting the rotation control neurons changed from \(-3.5\) to \(-5.0\).
Figure 5.19: The behaviour of the agent over 100 time units, returning from a beacon at \((0, 0.5)\), the rotation control neuron time constant is 0.005 (where the evolved value was 0.01). The length of the scale bar is 0.05 distance units.
Chapter 5. Two Dimensional Path Integration. I

118

Figure 5.20: The behaviour of the agent over 100 time units, returning from a beacon at \((0, 0.5)\), the rotation control neuron time constant is 0.02 (where the evolved value was 0.01).

Fig. 5.22 shows the agent’s behaviour when \(A = 0.5, B = 0.25\) distance units, and \(\alpha\) is varied from 0 to \(\pi\). The figure shows that the agent (diamonds) closely approximates the geometrically correct homing direction (solid line) and shows no tendency to mimic the errors observed in the ant under analogous conditions (squares). Fig. 5.23 shows the behaviour when \(A = 0.25, B = 0.5\), and Fig. 5.24 shows the case when \(A = 0.5\) and \(B = 0.5\). Again the agent follows the geometrically correct homing direction closely and does not have errors similar to the ant’s.

Comparative Topology Experiments

In order to test whether all redundancies had been removed from the network by the automatic pruning technique, four network topologies were defined and GA runs were performed to estimate the maximum fitness that could be obtained for each topology. Only four topology classes were necessary since it is known that all three sensor types (beacon, compass and speed) are required to solve the task. The only two weight pairs which therefore might be redundant are 5 and 6, since their removal does not directly or indirectly disconnect any sensor from the output neurons.

The four network topologies constructed were: class 1 the original network topology, class 2 the original minus weights \(w_{L5/R5}\), class 3 the original minus weights \(w_{L6/R6}\) and class 4 the original minus weights \(w_{L5/R5}\) and weights \(w_{L6/R6}\). These were used to re-evolve PI behaviour starting from weight values of zero in GA runs where the network topology was not allowed to mutate. Eight runs of 1500 generations were performed for each. The best agent in each of the final populations was tested in 1000 trials and scored for the number of times it reached the nest within the time limit. The best agent in each class was then evolved for a further 20000 generations, and
Chapter 5. Two Dimensional Path Integration. I

Figure 5.21: The behaviour of the agent over 100 time units, returning from a beacon at (0, 0.5), the rotation control neuron time constant is 0.04 (where the evolved value was 0.01).

Figure 5.22: The homing direction of the agent (diamonds) after visiting beacons marking out an L shaped journey with leg 1 of length 0.5 and leg two of length 0.25. The angle between the two legs is shown along the x axis. The homing direction (radians, clockwise from down) is shown along the y axis. The solid line shows the geometrically correct homing direction, the squares show the homing direction of *C. fortis* from Müller and Wehner (1988).
Figure 5.23: The homing direction of the agent (diamonds) after visiting beacons marking out an L shaped journey with leg 1 of length 0.25 and leg two of length 0.5. The angle between the two legs is shown along the x axis. The homing direction (radians, clockwise from down) is shown along the y axis. The solid line shows the geometrically correct homing direction, the squares show the homing direction of *C. fortis* from Müller and Wehner (1988).

Figure 5.24: The homing direction of the agent (diamonds) after visiting beacons marking out an L shaped journey with leg 1 of length 0.5 and leg two of length 0.5. The angle between the two legs is shown along the x axis. The homing direction (radians, clockwise from down) is shown along the y axis. The solid line shows the geometrically correct homing direction, the squares show the homing direction of *C. fortis* from Müller and Wehner (1988).
tested against the agent from 1500 generations. Only the class 4 agent showed an improvement, and was used in place of the generation 1500 agent. The generation 1500 agent was retained in the other classes. The four agents were then ranked for fitness. Each was tested six times for 1000 trials against the agents immediately above and below it, to determine if the ranking was statistically significant. The ranking order is class 1 > class 2 > class 3 > class 4. All differences were significant at $p < 0.01$ (Mann Whitney U test). Percentage success rates for classes 1 to 4 were 97.4, 89.1, 81.1 and 69.5 respectively over 6000 trials.

This supports the conclusion that no redundant weight pairs remained in the network after pruning (but cannot rule out the possibility that non-redundant weight pairs and/or nodes were pruned by mistake).

**Analytic Results**

The following results were obtained by a direct examination of the network topology and parameters.

**Phototaxis**

Phototaxis is achieved using a simple Braitenberg style (Braitenberg, 1984; Mittelstaedt and Eggert, 1989) architecture where each beacon sensor links to the same side rotation control neuron using a single fixed weight. The magnitude of these two weights (weight pair 1, see Fig. 5.8) is nearly always greater than that of weight pair 2 from the rest of the network. This means that while there is an active beacon anywhere in the arena the phototactic behaviour will automatically tend to suppress any other behaviour of the agent.

To see this consider that the maximum output of the beacon sensors is 1.0. Weighted by weight pair 1 this gives a maximum input to the rotation control neurons of 12.1, whereas the maximum input from weight pair 2 multiplied by the compass sensor output is approximately 11.0, calculated as follows. The maximum output of the compass sensors is magnitude 1.0. The maximum value of weight pair 2 is obtained when the agent begins a trial heading away from the nest at maximum speed for a distance of 1 unit (the furthest distance beacons are placed at). This is because input to weight pair 2 is from weight pair 3, which in turn is approximately a reactive copy of the speed sensor multiplied by weight pair 4. Weight pair 4 decays in magnitude over time starting from $-98.8$, therefore the beginning of the trial will yield the largest values. If we assume it approximately retained this value during the agent’s movement from the nest, maximum input to weight pair 2 would be given when the contralateral compass sensor output a value of magnitude $1$. Weight pair 2 decays in magnitude over time according to:

$$\dot{w}_2 = \frac{-w_2 - 98.8}{8.44}$$

giving $w_2$ a value, after one time unit, of (see Section B.3) $-11.0$. If the agent now turned such that the ipsilateral compass sensor output a value of magnitude 1.0 the input to the rotation control neuron would be maximised. Hence it appears that the magnitudes of these weights are adjusted such that phototaxis will virtually always dominate over any other turning tendency exhibited by the network while active beacons are present.
Figure 5.25: Part of the network of the agent evolved in experiment 2B which most directly implements the Mittelstaedt bicomponent PI and homing system. $C_L$ and $C_R$ are compass sensors, $S$ is the speed sensor, $R_L$ and $R_R$ are rotation control neurons. Here weight pair 2 is labelled as normal for the complete network (Fig. 5.8), weight pair 3 is labelled $kS$ and weight pair 4 is labelled $k$ for reasons explained in the main text.

**Implementation of the Mittelstaedt PI System**

Most of the rest of the network can be easily understood as an implementation of the Mittelstaedt bicomponent PI model in its simplest form (Eqns. 2.9). All network parameter values referred to below are given in Table 5.8. The part of the network shown in Fig. 5.25, consisting of compass and speed sensors, weight pairs 2, 3 and 4, and the rotation control neurons, approximates this model.

To see this relation we must first define a new variable $\lambda = \theta_A + \frac{\pi}{4}$ (where $\theta_A$ is the agent’s heading). The output of the compass sensors are now $C_L = \cos \lambda$ and $C_R = \sin \lambda$ and the effect is to rotate the x and y axes of the HV by $\frac{\pi}{4}$ radians. We will ignore for now the fact that the value of weight pair 4 decays from its initial value of $-98.8$ towards the value $-32.8$, and assume it has a constant value $k$. The time constant of weight pair 3, 0.0123, is very close to the smallest allowed during evolution, 0.01, and is such that weights $w_{L3}$ and $w_{R3}$ spend most of their time at the value $2.05 + kS$. Since $k$ is always much greater than 2.05, we will say that weight pair 3 approximates $kS$. The degree to which this relation is true depends on the relative rates of change of $S$ and weight pair 3. Since the forward motor neuron is always saturated on (having a large bias), the motor command tries to keep the agent at a constant speed of 1.0. Output motor noise and sensor noise are therefore the only sources of variation in $S$, motor noise having the larger magnitude (0.7 versus 0.01). Changes to noise are instantaneous, and hence weight pair 3 will take a little time to respond, but the Poisson process rate parameter for motor noise is low, 2.0, giving an average time between changes of 0.5 time units, meaning that weight pair 3 will spend most of its time at the value $kS$. Given this fact, the input to weights $w_{L2}$ and $w_{R2}$ will be approximately $kSC_R$ and $kSC_L$ respectively. Since the time constant on these weights is relatively large, 8.44, and they are initialised at zero (since ModCTRNN weights are initialised to their bias values), they act to approximately integrate their inputs.
Approximating Integration

The simplest way for a ModCTRNN weight (or CTRNN neuron) to act as an integrator is for it to have a large time constant relative to the time over which integration must occur. Consider the following two systems:

\[ \tau \dot{u} = I \]  
\[ \tau \dot{v} = -v + I \]

where \( \tau \) is a non-zero positive time constant, \( I \) is any constant real number. \( u \) and \( v \) are initialised to values \( u_0 \) and \( v_0 \) respectively at time \( t = 0 \). \( u \) is defined here as a perfect integrator of input \( I \), since its change since time 0 is proportional to the integral of \( I \) over this time. \( v \) is not a perfect integrator. The \( -v \) term makes its value approach the value \( I \) from its initial value by an exponential decay type process (see Section B.3). Treating \( v \) as an imperfect integrator in this fashion, the proportionate error of \( v \) with respect to \( u \) will be defined as:

\[ \varepsilon = \frac{(v - v_0) - (u - u_0)}{u - u_0} \]

which, taking \( v_0 = u_0 \), simplifies to:

\[ \varepsilon = \frac{v - u}{u - u_0} \]

The value of \( \varepsilon \) after time \( t \) has elapsed will be:

\[ \tau \left( 1 - \frac{u_0}{I} \right) \frac{1 - e^{-\frac{t}{\tau}}}{t} - 1 \]  

Implementation of the Mittelstaedt System Continued

Applying Eqn. 5.2 to the present case, we see that \( \tau = 8.44 \) and \( u_0 = 0.0 \). The latter means that the value of \( I \) is unimportant. We take the time over which the integrator must function to be approximately 2.0, the time taken to visit and return from a single beacon placed at a distance of 1.0 units from the nest, with the agent travelling at full speed. This gives \( \varepsilon = -0.11 \), or that the values of weight pair 2 would be around 10 per cent smaller in magnitude than a perfect integrator acting according to the definition of variable \( u \) (Eqn. 5.1). As will be seen below, another feature of the network in fact corrects for the decaying tendency of weight pair 2, and gives a higher accuracy than is implied by this simple analysis.

Given that weight pair 2 approximately integrates the terms proportional to \( S \sin \lambda \) and \( S \cos \lambda \), the two weight values constitute a geocentric Cartesian HV, since they implement the HV update equations of the Mittelstaedt PI model (Eqns. 2.9). The homing part of the same model is also implemented by the network as follows. The homing equation (expressed in terms of \( \lambda \), and using \( c \) as the turning rate coefficient to distinguish it from \( k \) used above) is:

\[ \dot{\lambda} = c (x \sin \lambda - y \cos \lambda) \]

Imagining for the moment that the rotation control neurons have a linear activation function, it
is easy to see that the remainder of the network implements this equation in a straightforward way. The time constant of the output neurons is very small, 0.0106, meaning that their internal states will be approximately equal to their inputs. The inputs of the left and right neurons are (in the absence of active beacons) \( w_{L2}\cos \lambda \) and \( w_{R2}\sin \lambda \). \( w_{L2} \) and \( w_{R2} \) act as the HV coordinates in this network and, since the rotation rate of the agent (\( \dot{\theta}_A = \dot{\lambda} \)) is proportional to the value of the left neuron minus that of the right, the network implements the Mittelstaedt homing system. This description is intended as a first approximation since, of course, we have neglected the influence of the neurons’ sigmoidal activation functions and the fact that the neurons do not respond immediately to changes in their inputs.

**Leakage Correction**

A first examination of the accuracy of integration of the HV values suggests we should expect errors on the order of 10 per cent, which seems anomalously large given the small size of the nest relative to the distances travelled by the agent. We now consider the fact that weight pair 4 decays during the course of each trial, which up to now we have considered as being constant. The decay of weight pair 4 follows:

\[
w_4 = -65.9304e^{-\frac{t}{5.1753}} - 32.8309
\]

such that for approximately the first 3 time units the decay of weight pair 4 approximates:

\[
w_4 = -98.7613e^{-\frac{t}{5.4355}}
\]

that is to say, it approximates a decay process with the same time constant as that of weight pair 2. This means that, during this period, the decay of weight pair 4 compensates almost perfectly for the decay occurring in the HV storing weights. The decay of weight pair 4 acts to continuously scale down the input to the HV integrators to match the degree of decay that the HV values have experienced since the start of the trial. This means that, although the HV values still decay, the ratio between the two values is still equal to the ratio that two non-decaying HV values would have. Appendix E shows how, if weight pair four decayed from its initial value towards zero with the same time constant as the HV integrator weights, it would provide a perfect correction for the decay of the HV. Of course, once weight pair 4 had reached a value of zero, input to the HV would also be zero. The HV itself would decay to zero, and the agent would cease turning (we could say that, having a zero HV, it now considered its current location to be the nest location regardless of its recent velocity). The actual behaviour of weight pair 4, which decays towards a value of \(-32.8309\), prevents this from occurring, at the expense of no longer functioning to accurately correct for HV decay towards the end of longer journeys.

The agent’s mean speed is 0.825 given a speed command of 1.0, with a random motor noise offset drawn from \([-0.7, 0.7]\), with the final value being clipped to a maximum of 1.0. The shortest possible complete journey is to one beacon at 0.5 units, giving a total distance of 1.0, and a mean time of 1.21. The longest possible journey is to three beacons placed at 1.0 units from the nest, the first and third being in the opposite direction to the second, giving a total (straight line) journey distance of 6.0 units, and a mean time of 7.27. A median journey would have two beacons at 0.75 units from the nest, separated by \( \frac{\pi}{2} \), giving a total distance of 2.56 and mean time of 3.10.
Hence approximately half of all journeys are expected to be completed, at least to the stage of nest search, within the time the leakage correction mechanism is operating with high accuracy. It seems possible that during the winding search pattern HV errors caused by decay would have a reduced tendency to accumulate since the agent’s motion is symmetrically arranged around the estimated nest location. Hence the gradual cessation of leakage correction after 3 time units appears to be a compromise allowing the agent to maintain a high enough turning rate at the end of long journeys at the expense of some systematic HV errors.

To summarise the implementation of the Mittelstaedt model in the evolved network, weight pair 2 provides a pair of Cartesian HV coordinates by integrating $k$ multiplied by the speed multiplied by the contralateral compass sensor. The leakiness of integration is corrected for by the exponential decay of $k$ over time which acts to compensate for the decaying memory of earlier movements.

**Output Neuron Filtering**

We expect a system closely following the Mittelstaedt model (but without forward speed control) to behave as described in Section 2.3.5 during homing, producing spirograph like patterns which appear unsuitable for efficient search behaviour, and which do not resemble the agent’s behaviour closely. We now augment the Mittelstaedt model by considering the filtering effect of the rotation control neurons, whereby their outputs do not respond immediately to input changes. For convenience we will continue to treat them as having linear activation functions. We will examine a three variable model describing the behaviour of the agent when its forward speed is set to zero but rotation is still allowed. This model is obtained by setting the two HV variables of the Mittelstaedt model to constant values (leaving the agent’s heading as a variable), and adding two new variables for the two rotation control neurons. This gives us a new three variable system. The next section will deal with the search patterns obtained when the agent is allowed to move forwards with output neuron filtering in effect.

Appendix G shows that the effect of output neuron filtering when the agent is stationary can be understood by analogy with a linearly damped pendulum such that the agent rotates as if it were a pendulum, where gravity comes from the direction of the estimated nest position with a force proportional to the distance from the nest. The contralateral weights connecting the two rotation control neurons act to alter the linear damping coefficient. Since the attraction to the homeward direction is weaker the closer the agent gets, this helps explain how the same system can produce straight homewards runs when starting from long range, but change into winding search patterns close to the nest.

**A Five Variable Model of the Searching Agent**

This section investigates a five variable model which is a simplified version of the full evolved network, whose aim is to reproduce the searching patterns seen in the agent using the smallest and simplest set of equations. The following model is used as the starting point:
\[ \dot{x} = \cos \theta \]
\[ \dot{y} = \sin \theta \]
\[ \dot{\theta} = 150[f(u) - f(v)] \]
\[ 0.01\dot{u} = -u + ky\cos \theta + wf(v) \]
\[ 0.01\dot{v} = -v + kx\sin \theta + wf(u) \]

where \((x, y)\) is both the agent’s actual location and its internal geocentric Cartesian home vector.

The model therefore assumes PI is perfectly accurate and that the agent moves at a constant speed of 1. \(\theta\) is the agent’s heading anticlockwise from the direction of the x axis, \(u\) is the internal state of the left rotation neuron and \(v\) that of the right. Since the full agent stores a scaled (as well as decaying) version of the HV, \(k\) scales the HV before it inputs to \(u\) and \(v\). We use \(k = -32.0\), based on the asymptotic value of \(w_d\). \(w\) is the contralateral weight between the two neurons, we use \(w = -3.52\) as the full agent has.

Since the HV is both accurate and non-decaying, we have removed any possible influence of the leakage correction mechanism on the search patterns. Since the mechanism operates on a relatively long time scale it seems unlikely to be pivotal in the production of the twisting search patterns.

\(f()\) is the activation function of the neurons. \(f(x) = \frac{1}{1 + e^{-x}}\) is the function used in the full agent. Here we use either \(f(x) = \frac{1}{1 + e^{-x}}\) or a linear function. We would like to approximate the sigmoidal function using linear activation functions if possible, since the sigmoid functions may not be essential for the search patterns. Observations of the searching agent show that the internal states of the neurons fall in the range \([-2.5, 0]\) during search. Fitting a linear function \((f(x) = Ax + B)\) to the sigmoid \((f(x) = \frac{1}{1 + e^{-x}})\) in this range using the least sum of squared error gives \(A = 0.1712, B = 0.4602\). All results below were produced using a Runge-Kutta 4th order numerical integration driver using a time step size of 0.0001.

The five variable model with sigmoidal functions reproduces the search patterns well (see Fig. 5.26), showing that the decaying of the HV is indeed not important for search.

Using the linear function with parameters \(A = 0.1712, B = 0.4602\), the agent’s behaviour also closely matches that of the full network (see Fig. 5.27), showing that sigmoidal output neurons are not required to make the search patterns.

The effects of varying the parameter \(A\) of the linear activation function were briefly investigated. A plot was produced for values of \(A\) ranging from 0.01 to 0.28 in increments of 0.01, where the agent was released from the location \((0, 0.5)\) and allowed to move around for 30 time units. Above a value of approximately \(\frac{1}{2} = 0.284\) the values of \(u\) and \(v\) tend to show divergent exponential growth, due to a positive feedback loop involving \(u\) and \(v\). Between 0.28 and 0.22 the agent approached the nest before entering into a figure of eight loop oriented with respect to, but not centred on the nest. Fig. 5.28 shows a 3D plot of the system in \((t, x, y)\) space, in this case with \(A = 0.26\), where it can be seen that the agent enters what appears to be a cyclic attractor after an initial transient. Fig. 5.29 show the attractor in \((x, y)\) space. At \(A = 0.21\) the agent falls into a more complicated, but still regular pattern (Fig. 5.30). At 0.2 the pattern is similar to the search pattern of the full network. At 0.19 the agent converged to a repetitive pattern (Fig. 5.31). From 0.18 to 0.14 the pattern is again similar to the irregular search pattern of the full network. At 0.13 an
Figure 5.26: A simplified five variable approximation of the full evolved network. The agent was released from the location (0,0.5) and allowed to home and search for 30 time units. The length of the bar is 0.05 distance units. The search pattern is centered on the nest location. This model used the sigmoidal function $f(x) = \frac{1}{1+e^{-x}}$ for the neuron activation functions. The search pattern is very similar to that produced by the full network.

Figure 5.27: A second simplified five variable approximation of the full evolved network. The agent was released from the location (0,0.5) and allowed to home and search for 30 time units. The length of the bar is 0.05 distance units. The search pattern is centered on the nest location. This model used the linear function $f(x) = Ax + B$ with $A = 0.1712, B = 0.4602$ for the neuron activation functions. The search pattern is again very similar to that produced by the full network, showing that sigmoidal functions are not required to produce it.
irregular search like pattern dominates except a window for around 5 time units beginning 15 time units after release when the figure of eight pattern is visible. For 0.12 the pattern shows a slightly irregular periodic pattern (Fig. 5.32). For $A = 0.11$ the pattern is irregular, but a new long thin looping pattern is visible towards the end. From $A = 0.10$ to 0.01 we see mainly approximately regular rotating patterns made up of long thin loops as show for the case of $A = 0.09$ (Fig. 5.33). The shape of the loops and rate of rotation and the degree of regularity varies according to the parameter value. Thus to a first approximation the model shows that the search patterns are only generated for a relatively narrow range of the parameter under study, much as was the case for the full network.

Adding long delays

*C. fortis* can retain some memory of its HV for long periods (up to about four days) if forcibly held still (Ziegler and Wehner, 1997). To test the agent’s ability to perform the same task with its leaky integrators we further evolved the network obtained in experiment 2B whilst subjecting
Figure 5.30: The linear activation function model, with parameter $A = 0.21$. The agent was released from $(0, 0.5)$. The scale bar is 0.02 units long. The pattern is centred on the nest location. This shows the agent’s motion from 45 to 50 time units after release.

Figure 5.31: The linear activation function model, with parameter $A = 0.19$. The agent was released from $(0, 0.5)$. The scale bar is 0.01 units long. The nest location is very close to the point where the trajectory crosses itself towards the bottom right of the picture. This shows the agent’s motion from 20 to 50 time units after release, and consists of multiple loops around the circuit shown.
Figure 5.32: The linear activation function model, with parameter $A = 0.12$. The agent was released from (0,0.5). This shows the agent’s motion from 0 to 30 time units after release.

Figure 5.33: The linear activation function model, with parameter $A = 0.09$. The agent was released from (0,0.5). This shows the agent’s motion from 0 to 60 time units after release.
the agent to a gradually increasing holding time at the last beacon. Finally, the agent was held for up to 50 time units (i.e. 50 times the longest possible direct return journey). The fittest agent returned to the nest within the time limit in 931 of 1000 test trials. The network had evolved to retain the same structure but had increased the time constant on the HV storing weights $w_{L2/R2}$ from 8.44 to 113.8, and had also increased the magnitude of weights $w_{LA/R4}$ from approximately 100 to 153 (this was achieved by adding a new pair of weights with the same start and end points, since the maximum allowed strength of any single weight was set to $\pm 100$). This modification is understandable since increasing the time constant of the integrators leads to less leakage (and so approximates perfect integration more closely) but smaller HV values. Smaller values in turn reduce the agent’s maximum rate of turn during homing, which influences the straightness of its path under noisy conditions, and the shape of its search pattern. Increasing the magnitude of $w_{LA/R4}$ compensates by scaling the HV values back up.

Varying the compass sensor

All the remaining experiments consist of applying variations to the shape of the compass sensor response function. This was felt to be an important assumption to study since the ant’s response function is not known in any detail (but see Labhart, 2000), and the evolved solution may clearly be heavily influenced by the choice made.

5.3.5 Experiment 3A: ModCTRNN constant speed with linear cosine

The fittest agent used two pairs of compass sensors with maximum responses set to approximately $\pm \pi/4$ and $\pm 3\pi/4$, but, since responses of sensors separated by $\pi$ radians are equal in magnitude but opposite in sign, this was equivalent to a single pair of sensors at $\pm \pi/4$. Taking this into account, the network had the same structure as that evolved in experiment 2B except it lacked weight pairs 4, 5 and 6 (see Fig. 5.8). The agent returned to the nest in 399 of 1000 random independent trials.

5.3.6 Experiment 3B: ModCTRNN variable speed with linear cosine

The fittest agent returned to the nest in only 82 of 1000 test trials. The agent used one compass sensor pair with maximum response angles of approximately $\pm \pi/4$. The network had the same architecture as experiment 3A, i.e. it did not take into account information from the speed sensor, even though speed was not constant in this experiment. This accounts for its much poorer performance.

5.3.7 Experiment 4: ModCTRNN constant speed with head direction cells

The fittest agent returned to the nest in only 56 of 1000 test trials. The agent used three pairs of compass sensors with maximum response angles set to approximately $\pm \pi/4$, $\pm 7\pi/12$ and $\pm 3\pi/4$. The network structure was very different from those evolved in the other experiments but, since the performance was so poor, it cannot be considered a viable PI model and will not be presented here.
5.3.8 Experiment 5: ModCTRNN constant speed with positive cosine

This experiment failed to produce any successful agents.

5.4 Discussion

GeNGA successfully evolved a bicomponent model of PI using very few neurons and weights provided the ModCTRNN model was used. The bicomponent architecture is probably the simplest implementation of PI and was therefore easier for the GA to discover, and is modular in that the two components of the HV do not have to interact in order to be correctly updated. The imposition of bilateral symmetry and compass input and motor output based on pairs of sensors and neurons probably also encouraged the production of a bicomponent system. Only cosine shaped compass response functions evolved successful solutions, probably because this allows the sensor output to be fed almost directly into the HV. Wittmann and Schwegler (1995) show that any symmetrical unimodal compass response function can be converted into a sinusoidal shape, but this requires extra processing between the sensors and the HV accumulator which our experiments using non-cosine compasses failed to evolve, although this could be due to the small number of replicates performed. The compass representation we chose has negative as well as positive values, which clearly cannot be interpreted directly as a firing rate. However each compass sensor could be replaced with two sensors, each having only positive outputs, one representing the positive half of the cosine shape, the other the negative part. This solution closely resembles the stimulus-response behaviour of four sensory interneurons of the cricket cercal sensory system (Miller et al., 1991), which show half-wave rectified sine waves. The evolved network is then easily modified by the addition of further weights and would retain the same behaviour. The network then requires approximately 30 weights in place of the original 12.

The model automatically generates search behaviour at the nest that is similar in some respects to the behaviour of the ant. This was unexpected, and not the result of any explicit feature of the artificial selection scheme. It seems likely that the Wittmann-Schwegler and Hartmann-Wehner models could produce a similar behaviour, but the authors never present any analysis of the dynamics of their models during homing. Hartmann and Wehner (1995) assume systematic search requires a separate control system which the HV simply triggers when needed. Here we have shown that such an extra mechanism maybe entirely unnecessary. The agent’s search pattern produces a bell shaped density profile similar to that of *Cataglyphis*, although the ant’s pattern is more adaptable than our model’s, since the ant searches a wider area the further from home its journey took it, and during individual searches progressively searches a wider area over time (Wehner and Srinivasan, 1981; Wehner, 1992).

The evolved solution to experiment 2B was essentially the geocentric Cartesian Mittelstaedt bicomponent model, with leakage correction and searching added. The searching behaviour appeared to be tuned to a region of parameter space which generated dense, space-filling and non-repetitive trajectories centred symmetrically on the nest, analogous to the ant’s own search patterns.
Chapter 6

Two Dimensional Path Integration. II

6.1 Overview

The navigation task presented in this chapter differs from the previous chapter’s in that the agent has rotation rate sensors instead of compass sensors. A hand built CiTRuS network is presented which performs the task using the simplest type of HV with these sensors, namely the egocentric Cartesian form. The features of leakage correction and filtering-generated search patterns, identified in the evolved controller in the previous chapter, are converted for use with an egocentric Cartesian PI system. In agreement with existing work relating to sensor errors in PI, the CiTRuS model is found to be more sensitive to errors in the rotation rate sensors than the previous chapter’s network was to errors in its compass sensors.

6.2 A CiTRuS Controller for PI Using Rotation Rate Sensors

Fig. 6.1 shows the network topology of a CiTRuS controller designed to perform PI using an egocentric Cartesian HV. Tables 6.1 and 6.2 show the parameter values used. The network was designed to follow the 2B (evolved geocentric Cartesian) network as closely as possible, such that the HV components have similar magnitudes and identical decay rates to the earlier network. The sign of the $W$ leakage correction weight has been changed to positive, and the definition of the agent’s rotation rate changed to $\dot{\theta} = 150(R_R - R_L)$ to make a comparison with standard differential steer robots more straightforward. The interpretation is now that the rotation actuators on the left and right side perform pushing rather than pulling actions. This means that positive weights from the beacon sensors must now be connected contralaterally to the rotation control neurons in order to give positive phototaxis (this is the more usual arrangement, see for instance Mittelstaedt and Eggert (1989)).

6.2.1 The Design of the Agent

Eqn. 2.10 shows my symmetrical version of the canonical egocentric Cartesian HV update equations. These are the basis for the bilaterally symmetrical control network. First we convert the leakage corrected form of egocentric Cartesian PI (Eqns. E.7 and E.8) into symmetric form:
\[
\alpha \dot{u}' = -u' + \alpha \theta v' - \frac{w_0}{\sqrt{2}} \\
\alpha \dot{v}' = -v' - \alpha \theta u' - \frac{w_0}{\sqrt{2}}
\]  

(6.1)

where \( \alpha \dot{w} = -w \) with \( w_0 = \alpha k \) is the decaying leakage correction value. This implies we require two leaky integrator neurons to hold \( u' \) and \( v' \) (these are labelled simply as \( X \) and \( Y \) in Fig. 6.1 to make it clear they store the HV), each receiving two inputs: one from the speed sensor scaled by the decay value \( w \), another from the contralateral HV component multiplied by the ipsilateral rotation rate sensor. The output to the rotation neurons needed to implement the rotation control equation of Eqns. E.8 \((\dot{\theta} = k \sqrt{\frac{1}{2}} (Y - X))\) is simply to input \( u' \) to one side and \( v' \) to the other with the correct weighting.

The parameter values are set to mimic or be equivalent to the corresponding parameter from the evolved geocentric Cartesian HV network from experiment 2B (see Table 5.2 and Fig. 5.8). The CiTRuS node parameters of neurons \( R_{L/R} \) (the rotation rate control neurons) are set to give a sigmoid output identical to a CTRNN node (with a minimum output of 0 and maximum of 1). The bias-like \( d \) parameter is equal to the bias of the rotation control neurons in the 2B network and the value of \( g \) is set to the reciprocal of the time constant from the 2B network. The forward speed control neuron is saturated to output 1, as with the 2B network. The bias term \((d)\) and initial value \((v_0)\) of neuron \( W \) ensure that the output begins at 98.76 and decays towards 32.83, in equivalence with the leakage correction value \( w_{L/R4} \) from network 2B. The sigmoid function parameters \( a, b \) and \( c \) of neurons \( X, Y \) and \( W \) are set to act as identity functions, mapping internal states to identical output value, given internal states within the ranges of approximately \([-100, 100]\) (for \( X \) and \( Y \)) and \([-250, 250]\) (for \( W \)). This simulates the behaviour of the non-sigmoidal ModCTRNN weights from network 2B. The \( g \) values of \( X, Y \) and \( W \) are the reciprocals of the \( \alpha \) time constants of 2B network weights \( w_{L/R2} \) and \( w_{L/R4} \) respectively. Weights \( B_{L/R} \rightarrow R_{R/L} \) and \( R_{L/R} \rightarrow R_{R/L} \) are identical to their 2B counterparts. Weight \( Sp \cdot W \rightarrow X/Y \) provides the \(-\frac{1}{\sqrt{2}}\) coefficient from Eqn. 6.1. Weight \( X/Y \cdot C_{R/L} \rightarrow X/Y \) is set to \( \frac{334}{750} \) to convert the normalised rotation rate \((\in [-1, 1])\) back into \( \theta \) and multiply it by \( \alpha \).

### 6.2.2 Behaviour of the CiTRuS Agent

The behaviour is very similar to that of the evolved agent from the previous chapter when tested with identical noise conditions. The agent performs phototaxis and then homes in a roughly straight line. A search pattern is generated at the estimated nest location, which lasts until the trial ends (Fig. 6.2). The most noticeable difference, apart from the obvious difference in the behaviour of the HV (Fig. 6.3) is an increase in the effect of noise added to sensors \( C_{L/R} \): the agent’s estimation of the nest location is significantly less accurate in this system for the same noise magnitude (see below). This is as expected (Benhamou et al., 1990) since these sensors now provide an idiothetic compass, which is subject to the accumulation of error itself, before we consider HV error accumulation. To see this, we can imagine a system which operates using rotation rate sensors \( (C_{L/R}) \) but which uses a geocentric Cartesian HV \((x,y)\) internally. To update the HV we require \( s \cos \theta \) and \( s \sin \theta \). To estimate \( \theta \) we can integrate the average of the two rotation sensors: \( \dot{\theta} = \frac{1}{2} \int C_L - C_R dt \), and integrate \( s \cos \dot{\theta} \) and \( s \sin \dot{\theta} \). Information from the rotation sensors is integrated twice, meaning that a transient error in the sensor values gives a constantly increasing error.
Table 6.1: Node parameter values of the CiTRuS PI system using rotation rate sensors. Nodes are: \( R_{L/R} \), left/right rotation rate control neurons; \( F_w \), forward speed control neuron; \( X/Y \), egocentric Cartesian HV neurons; \( W \), decaying leakage correction value. The \( g \) parameter indicates the response rate of the node, and is achieved by setting the \( g \)-function parameters as follows: \( a^g = b^g = g, c^g = d^g = 0 \) (the variable response rate feature of CiTRuS is thus not utilised here).

<table>
<thead>
<tr>
<th>Node</th>
<th>( v_0 )</th>
<th>( g )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( d )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_{L/R} )</td>
<td>0</td>
<td>94</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0.3</td>
</tr>
<tr>
<td>( F_w )</td>
<td>0</td>
<td>100</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( X/Y )</td>
<td>0</td>
<td>0.12</td>
<td>400</td>
<td>-400</td>
<td>0.005</td>
<td>0</td>
</tr>
<tr>
<td>( W )</td>
<td>65.93</td>
<td>0.19</td>
<td>1000</td>
<td>-1000</td>
<td>0.002</td>
<td>32.83</td>
</tr>
</tbody>
</table>

Table 6.2: \( w \) weight values for the CiTRuS egocentric PI network. All \( \phi \) weights are unity. The weight name indicates the local network topology leading into and out of the weight. The network is bilaterally symmetrical. So, for example, \( X/Y \cdot C_{R/L} \rightarrow Y/X \) indicates a \( w \) weight into neuron \( Y \) from a multiplier (m-)node outputting the product of \( X \) and \( C_R \). The contralateral \( w \) weight, sharing the same magnitude, inputs to neuron \( X \) from an m-node outputting the product of \( Y \) and \( C_L \).

<table>
<thead>
<tr>
<th>Weight</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B_{L/R} \rightarrow R_{R/L} )</td>
<td>12.07</td>
</tr>
<tr>
<td>( R_{L/R} \rightarrow R_{R/L} )</td>
<td>-3.52</td>
</tr>
<tr>
<td>( X/Y \rightarrow R_{L/R} )</td>
<td>1</td>
</tr>
<tr>
<td>( X/Y \cdot C_{R/L} \rightarrow Y/X )</td>
<td>1266</td>
</tr>
<tr>
<td>( S_p \cdot W \rightarrow X/Y )</td>
<td>-0.707</td>
</tr>
</tbody>
</table>

Figure 6.1: CiTRuS network performing the PI task using rotation rate sensors and an egocentric Cartesian HV. A leakage-corrected leaky HV is employed. The rotation control neurons act to filter the output of the Mittelstaedt homing system to produce search patterns in the same manner as the evolved PI network with compass sensors. \( B_{L/R} \), left/right beacon sensor; \( C_{L/R} \), left/right rotation rate sensors; \( S_p \), forward speed sensor; \( R_{L/R} \), left/right rotation control neuron; \( X/Y \), egocentric Cartesian HV; \( W \), decaying leakage correction value. The bar across the beacon sensor-to-rotation neuron weights indicates that the weight (drawn as if ipsilateral for convenience) is in fact contralateral. Small black dots are CiTRuS m-nodes (which multiply their inputs together).
Figure 6.2: Motion of the idiothetic compass agent as it visits three beacons (not shown) and homes using a leaky egocentric Cartesian home vector. The agent starts from the nest at \((0, 0)\), but centres its search at approximately \((-0.14, 0.11)\).

Figure 6.3: Dynamics of the egocentric Cartesian HV storing neurons during the journey shown in Fig. 6.2. Dotted line, internal state of neuron X in Fig. 6.1; continuous line, internal state of neuron Y in Fig. 6.1.
Figure 6.4: Average homing error of the idiothetic- (Fig. 6.1) and allothetic-compass (Fig. 5.8) networks. Homing error is defined as the average distance of the agent from the nest during the last 5 time units of a 15 time unit excursion to between 1 and 3 randomly placed beacons (see Section 5.2.2). Plots are averages taken over 100 trials. Noise magnitude is as defined for the evolution of the allothetic-compass network (see Section 5.2.4). Pluses, noise applied to rotation rate sensors of idiothetic network; Crosses, noise applied to speed sensor of idiothetic network; Asterisks, noise applied to compass sensors of allothetic network; Squares, noise applied to speed sensor of allothetic network. Noise applied to the non-varied sensors was fixed at magnitude 0.01. At and above noise magnitude 0.04 for the idiothetic-compass sensors, the idiothetic-network is no longer able to perform phototaxis to the beacons.

in the HV, since the value of $\hat{\theta}$ acts as a memory of the transient error. Transient errors in allothetic compass sensors would lead only to corresponding transient errors in the updating of the HV.

A direct comparison between the evolved allothetic compass agent and the current hand designed idiothetic compass agent is not entirely fair, since the idiothetic agent has not been tuned through thousands of generations of mutation and selection. It is, however, designed to perform the task in an equivalent way to the evolved network, using the fact that geocentric Cartesian and egocentric Cartesian PI are alternative descriptions of essentially the same process. Fig. 6.4 shows the effect of varying the magnitude of sensor errors for the two networks, for both the speed and compass/rotation sensors. The idiothetic network shows less accurate homing for any given level of sensor noise, and a much greater reduction in accuracy in response to increases in noise applied to the rotation rate sensors. For the idiothetic network’s speed sensor the homing error does not varying significantly as noise is increased from 0 to 0.2, but rather reflects the 0.01 magnitude noise applied to the rotation sensors.

6.3 Conclusions

The CiTRuS agent demonstrates that PI and searching can be performed using a leakage-corrected egocentric Cartesian HV. A fuller investigation would be required to compare it’s behaviour with that of the evolved allothetic-compass agent thoroughly, but the results presented here are in agree-
ment with previous work on the effect of errors on PI accuracy (Benhamou et al., 1990), namely that an idiothetic compass renders the system much more vulnerable to errors in the corresponding sensors (i.e. the rotation rate sensors).
Chapter 7

Vector Navigation

7.1 Overview

A hand built CiTRuS controller is presented capable of performing a “vector navigation” task. Here vector navigation (Collett et al., 1999) is used to denote navigation based on PI, but involving more complex behaviour than simply returning to the nest after a single excursion. The task involves the agent remembering the location of the last beacon it visited prior to homing to the nest. Upon reaching home the agent must perform shuttle runs between the last beacon and the nest, without the aid of phototaxis to relocate the beacon. Hence it must be able to memorise the location of the beacon and return there using PI in a new mode of operation, namely that of navigating to a location other than the nest. The system is built on a generalisation of the first evolved ModCTRNN PI controller (see Chapter 5) and requires the latching feature of the CiTRuS controller. The basic mechanism employed follows that used by Mittelstaedt and Mittelstaedt (1973), whereby the coordinates of a memorised location are subtracted as offsets from the current HV, before the HV is fed to the homing subsystem. In this way navigation to the location proceeds in exactly the same way as homing to the nest, in that the agent is directed towards the location where the vector input to the homing system is zero.

Desert ants can return to a familiar feeding location using PI alone but seem unable to remember an inbound and an outbound journey which are not the reverse of each other (Collett et al., 1999; Wehner et al., 2002), and have also been shown to reset (zero) their HV when they enter their nest Knaden and Wehner (2006). The present model incorporates both of these features.

The model is presented to demonstrate the use of CiTRuS latching in the context of a biologically relevant task (spatial memory). It also illustrates how multiplier nodes allow the operation of multiplication to be carried out directly without requiring the method of ModCTRNN, where a weight variable with low time constant acts to approximate a multiplier node. This incurred the cost of an additional variable for each multiplication performed and therefore increased the complexity. ModCTRNN also required weights to be chained together to multiply more than two variables, further increasing the total number of variables required. Finally, the vector navigation model illustrates how the mechanisms uncovered by analysis of the earlier evolved PI system can be applied in this more complex situation. Here we apply the features of leakage correction and
output filtering-mediated search patterns from the earlier PI system. It is shown that leakage correction and search patterns are compatible with a system which stores vectors for later recall. It remains an open question whether a GA could construct an equivalent CiTRuS vector navigation system by itself, and whether it would resemble the one presented here.

7.2 Sketch of the Vector Navigation System

Mittelstaedt and Mittelstaedt (1973) show a method for generalising the basic PI based homing system, whereby a pair of coordinates representing the location of a prey item can be stored and used to return there later. This is simplest to achieve in a geocentric system (see Section 2.2.4) since geocentric but not egocentric HVs contain sufficient information to allow the animal to return to the same location at a later time. An egocentric system can achieve this only if the compass heading is stored in addition to the HV at the target location.

The vector navigation agent contains two extra sensors on top of those required by the evolved PI agent, namely a nest and a food sensor. The nest sensor is active only when the agent is within the circle defined as the nest location, and similarly the food sensor is only active inside the circle defined as the location of the last beacon. Here the last beacon is also considered to be the location of a source of food to which the agent must return as many times as possible. The other sensors, and the motor control neurons are as before: two beacon sensors, two allothetic compass sensors, one speed sensor, one forward speed control neuron and two rotation control neurons.

The navigation system contains two vectors: the home vector (HV) and the stored vector (SV), both are geocentric Cartesian vectors consisting of two scalar values. The HV is continuously updated as per the evolved ModCTRNN system. Phototaxis to between one and three randomly placed beacons is used to generate the agent’s initial random excursion as before (see Chapter 5). Homing is suppressed during phototaxis also using the same method as the evolved network, by giving the phototactic weights a larger magnitude than those of the homing system. The homing system receives input such that it sees either the HV or the HV minus the SV, and thus directs the agent to the location where \( \text{HV} = (0, 0) \) or to where \( \text{HV} - \text{SV} = (0, 0) \). A switch neuron controls which input the homing system receives. Transient activation of the nest sensor activates the switch, and likewise transient food sensor activation deactivates the switch. Therefore when the switch is active and no beacons are visible the agent is directed towards the location indicated by the SV. Nest sensor activation also causes the HV to be zeroed, and food sensor activation causes the HV to be copied to the SV. Therefore upon reaching the final beacon (the feeder location) the HV is copied to the SV and the switch deactivates. The homing system now takes the agent to the nest location. Upon reaching the nest the HV is zeroed, and the switch is activated. Now the “homing” system takes the agent back to the beacon. The agent should then continue to move back and forth between the nest and the final beacon. Since the homing system behaves in the same manner during homing and returning to the feeder, it is simple to generate the same kind of search patterns both at the nest and feeder, by simply adding the output filtering system. Both HV and SV are stored on leaky integrators. The time constant of decay is increased relative to the evolved PI system since the total time taken to complete multiple trips to the feeder is longer. The leakage correction system is implemented in the same way as before, where the input to the HV is scaled down by a decaying coefficient which initially matches the decay profile of the HV.
itself, but which levels off at a non-zero value to ensure the agent continues to respond. The time constant of the SV simply has to match that of the HV. The complete vector navigation system is defined in detail below.

It is interesting to note the similarity of the SV in this network to the memory cell structure of long short-term memory (LSTM) networks (Hochreiter and Schmidhuber, 1997). Although the context and aim are rather different, since LSTM networks are discrete time recurrent networks trained by a back propagation-like method, the operation is similar. In LSTM multiplier nodes are used to gate the input and output of memory cells to protect, respectively, the internal state from corruption by irrelevant inputs and other neurons from disruption by inappropriate output from the memory cell. This is similar to the way that, in the present model, output from the SV is gated (using an m-node) by the switch neuron (see below) and input is (m-node) gated by the food sensor. In LSTM networks this structure is hardwired into each memory cell, but in CiTRuS it would be available for exploitation by evolution.

7.3 Detailed Definition of the Vector Navigation System

Table 7.1 gives the parameter values of the network nodes, Table 7.2 gives the $w$ weight values (all $\phi$ weights are set to 1) and Fig. 7.1 shows the topology. Two alterations were made to the definition of the agent relative to that used for the evolved PI system. Firstly the left and right compass sensors were defined to output $\cos \theta$ and $\sin \theta$ respectively rather than $\cos(\theta + \frac{\pi}{4})$ and $\cos(\theta - \frac{\pi}{4})$, such that the x and y axes of the home vector coincide with those of the simulated arena. Secondly the definition of the agent’s rotation rate was defined as $\dot{\theta} = 150(R_R - R_L)$ instead of $\dot{\theta} = 150(R_L - R_R)$, to make the system more comparable with a differential steer robot.

Below “2B network” is used to refer to the evolved ModCTRNN PI system presented in Section 5.3.4. The beacon sensors are connected to the rotation control neurons with contralateral weights, of magnitude 12.07, where the 2B network had ipsilateral weights of the same magnitude. The rotation neurons are connected together by mutual contralateral weights of magnitude $-3.52$, also as in the 2B network. The CiTRuS parameters of the rotation control neurons are defined to mimic the same neurons of the 2B network (recall that the output of CiTRuS nodes is a sigmoidal function whose maximum and minimum are set by parameters $a$ and $b$ respectively, see Section 3.2.3). The forward motor neuron’s output is 1.0 unless the nest or food sensor are active, when it outputs 0.05. This slows the agent down to ensure it has time to zero the HV or store the SV at the nest and food respectively. The nest and food sensors also control the value of the switch neuron. Both cause the response rate of the neuron to increase to maximum, 100, and the nest sensor also inputs to the $I$ term, meaning that activation of either sensor causes the switch neuron to latch to the current output of the nest sensor, giving the desired switch behaviour (activation at the nest and deactivation at the food).

The decay rate of the HV and SV neurons is reduced from that of the 2B network in order to allow PI to function for the duration of multiple trips to the food, where the 2B agent only made one round trip. The decay rate, 0.02, is equivalent to a ModCTRNN $\tau$ value of 50, where the 2B network value was 8.44, giving the present network a memory duration approximately 6 times as great. The response rate (i.e. the output of the $g$ function) of the HV neurons is equal to 0.02 when the nest sensor is inactive, giving the agent a leaky HV analogous to the 2B network while
Table 7.1: CiTRuS neuron parameters used for the vector navigation network (weight values are shown in Table 7.2). $R_{L/R}$, left/right rotation control neurons; $F_w$, forward speed control neuron; $X/Y$, HV integrator neurons; $S/T$, SV neurons; $S_w$, switch neuron; $F_l$ nest sensor filter neuron; $N_o$, nest sensor not neuron; $W$, leakage correction decay value neuron.

<table>
<thead>
<tr>
<th>Node</th>
<th>$v_0$</th>
<th>$a^g$</th>
<th>$b^g$</th>
<th>$c^g$</th>
<th>$d^g$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_{L/R}$</td>
<td>0</td>
<td>94</td>
<td>94</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0.2994</td>
</tr>
<tr>
<td>$F_w$</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.05</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>$X/Y$</td>
<td>0</td>
<td>100</td>
<td>0.02</td>
<td>1</td>
<td>-10</td>
<td>400</td>
<td>-400</td>
<td>0.005</td>
<td>0</td>
</tr>
<tr>
<td>$S/T$</td>
<td>0</td>
<td>100</td>
<td>0.02</td>
<td>1</td>
<td>-10</td>
<td>400</td>
<td>-400</td>
<td>0.005</td>
<td>0</td>
</tr>
<tr>
<td>$S_w$</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>1</td>
<td>-10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-10</td>
</tr>
<tr>
<td>$N_o$</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>$W$</td>
<td>0.68</td>
<td>0.022</td>
<td>0.022</td>
<td>0</td>
<td>0</td>
<td>1000</td>
<td>-1000</td>
<td>1</td>
<td>0.12</td>
</tr>
</tbody>
</table>

Table 7.2: $w$ weight values of the CiTRuS vector navigation system. All $\phi$ weights are set to 1. $\rightarrow$ and $\Rightarrow$ indicate input to the $I$ term and $I^g$ term of the receiving neuron respectively. Dots (·) indicate values multiplied together at a multiplier node before being weighted and inputted to the receiving neuron. $F_w, R_{L/R}, X/Y, S/T, F_l, N_o, S_w, W$ are neurons as defined in Table 7.1. Sensor names are as follows: $B_{L/R}$, left/right beacon sensor; $C_{L/R}$, left/right compass sensor; $S_p$, speed sensor; $F_d$, food sensor; $N_e$, nest sensor.
it is away from the nest. At the nest the HV is rapidly zeroed using carefully controlled changes to the neuron response rates and \( I \) inputs, mediated by the \( No \) (not) and \( Fl \) (filter) neurons. The nest sensor acts via these neurons to both zero input to the HV integrators and to increase their response rates. The \( No \) neuron is required to invert the output of the nest sensor such that a zero is available at the nest to multiply the HV input by. Since the nest sensor changes value instantaneously and since the HV \( g \) function output also responds instantly to its input it would be possible for the neuron response rates to become maximal before the not neuron had become zero. Hence a filter neuron is required to delay the increase in HV response rate. The parameters of the not and filter neurons are such that the not neuron output has reached zero before the filter neuron output has begun to increase. This prevents the HVs transiently responding to non-zeroed inputs at an elevated response rate. The other inputs to the HV are as for the 2B network: the compass sensor outputs are multiplied by the speed sensor output and by the decaying leakage correction coefficient.

The response rate of the SV neurons is increased by the food sensor to cause them to latch to the current value of the HV. Away from the food the SV \( I \) term input is held at zero by the inactivity of the food sensor and the response rate returns to that of the HV decay rate such that the SV decays at the same rate as the HV. SV input must be zeroed away from the food to prevent integration of the \( I \) term input from the HV.

Navigation to the nest results from inactivation of the beacon sensors (when the final beacon is reached and extinguished) and of the switch neuron as occurs at the feeder. The switch now suppresses the input to the rotation control neurons from the SV neurons, leaving only the input from the HV multiplied by the contralateral compass sensors, just as for the 2B network. Upon switch activation at the nest, input now reaches the rotation motor neurons from the SV, again multiplied by the contralateral compass sensor. This is the offset mechanism of Mittelstaedt and Mittelstaedt (1973). The negative \( w \) weights from the SV to the motor neurons acts to subtract the SV from the HV, causing motion towards the point where the HV equals the SV. Since the settings of the rotation neurons are the same as for the 2B network, they act to filter the output of the Mittelstaedt homing system, and so can give rise to search patterns at both the expected nest and food locations.

### 7.4 Behaviour of the Vector Navigation System

The agent was tested in 1000 trials each of 200 time units with all sensory and motor noise removed. From one to three randomly placed beacons were present for each trial, placed at random bearings from the nest at distances of between 0.75 and 1.0 distance units. Fig. 7.2 shows the frequency distribution of the number of round trips the agent completed per trial. Fig. 7.3 shows the same graph for 1000 trials performed with noise present as for the evolution of the 2B network: forward motor noise magnitude 0.7, rate 2.0; rotation neuron noise magnitude 0.1, rate 20.0; all sensor noise magnitudes 0.01, rates 20.0. Figs 7.4 and 7.5 show example trajectories of the agent under these two condition.

An interesting feature of the agent’s search behaviour can be seen clearly in the trajectory lacking noise. The agent tends to overshoot the nest or feeder if it does not find it on the first pass. At least with motor noise removed, the agent will on average travel further than the HV
Figure 7.1: CiTRuS network topology used to implement the vector navigation system. The complete network is bilaterally symmetrical, and is shown here with each bilateral pair of components collapsed down to a single symbol. The bilateral components are: $B_{L/R}$, left/right beacon sensors; $C_{L/R}$, left/right compass sensors (outputting $\cos \theta$ and $\sin \theta$ respectively); $X/Y$, HV integrators; $S/T$, SV (stored vector) neurons; $R_{L/R}$, left/right rotation control neurons. Non-bilateral components are: $Fd$, food sensor; $Ne$, nest sensor; $Sp$, speed sensor; $Fw$, forward speed control neuron; $W$, decaying leakage correction coefficient; $No$, nest sensor not neuron; $Fl$, nest sensor filter neuron; $Sw$, offset mechanism switch neuron. Small black circles are multiplier (m-) nodes, single headed arrows are $\phi$ weights into m-nodes or $w$ weights into neuron $I$ terms, double headed arrows are $w$ weights into neuron $I^s$ terms. Short lines drawn immediately behind arrow heads indicate that the weight is contralateral rather than ipsilateral. For example the connection from $B_{L/R}$ to $R_{L/R}$ represents: an ipsilateral $\phi$ weight from $B_L$ to an m-node, followed by a contralateral $w$ weight from the m-node to $R_R$; and an ipsilateral $\phi$ weight from $B_R$ to another m-node, followed by a contralateral $w$ weight from this m-node to $R_L$. 
Chapter 7. Vector Navigation

Figure 7.2: Frequency distribution of the number of trips from the nest to the food and back completed by the agent during 200 time units, with all sensor and motor noise removed. 1000 trials were performed in total. Between one and three beacons were placed at random bearings from the nest at a distance of between 0.75 and 1.0 distance units.

or SV indicates it should before making its first turn into a search pattern. This counteracts the tendance of the agent to travel too short a distance due to the decay of the HV (see Section 2.3.6). Additionally we note that if the agent, after completing the phototactic part of the trial, spends most of its time travelling back and forth along the line connecting the nest and feeder, then its heading will always be directed towards the nest or directly away from it and hence HV decay will have the property of only inducing errors to accumulate in the distance component, not the angular. Thus in this situation the expected major navigational error arising from integrator leak is automatically corrected by a feature of the dynamics of homing, namely undershoot is counteracted by overshoot. Since the agent zeroes the HV everytime it returns home and restores the SV each time it finds the feeder, its search behaviour only needs to compensate for the errors which have arisen during the current round trip. Of course, as can be seen in the trajectory of the noisy agent, should a single search episode last for a long time before the goal is located, then the HV and SV will have decayed significantly more, and the distance to the subsequent goal (whether nest or feeder) may be grossly underestimated. Hence in Fig. 7.4, a long search for the nest precedes a subsequent unsuccessful and misplaced search for the feeder.

7.5 Conclusions

The viability of a vector navigation system implemented using a CiTRuS controller has been briefly demonstrated. The system is known to function under high levels of motor noise and modest levels of sensor noise, equivalent to those of the evolved ModCTRNN PI system. The two novel features of that system have been successfully transferred into the present system, namely the presence of a partially corrected decaying HV and of output filtering-mediated search patterns. Leaky integrators were found to be compatible with a HV storage system (the SV), provided the
Figure 7.3: Frequency distribution of the number of trips from the nest to the food and back completed by the agent during 200 time units, with sensor and motor noise as used during the evolution of the 2B network: forward motor noise magnitude 0.7, rate 2.0; rotation neuron noise magnitude 0.1, rate 20.0; all sensor noise magnitudes 0.01, rates 20.0. 1000 trials were performed in total. Between one and three beacons were placed at random bearings from the nest at a distance of between 0.75 and 1.0 distance units.

Figure 7.4: Example trajectory of the agent with noise present. The agent completes 14 round trips before getting lost. After reaching the single beacon marking the feeder by phototaxis, the agent spends the first approximately 60 time units making shuttle runs between nest and feeder. Brief periods of search like behaviour can be seen at both the nest and feeder locations. Between approximately times 60 and 90 the agent is searching for the nest for an extended period. After finding it around time 90 it moves towards to feeder, but underestimates the distance significantly, and spends the remainder of the trial searching for it approximately two-thirds of the way from the nest to the true feeder location.
Figure 7.5: Example trajectory of the agent without noise. The agent completes 72 round trips during the 200 time unit trial, and does not get lost. Only the first 40 time units are shown. After visiting the two beacons, the agent travels towards the nest which it narrowly misses and overshoots. The nest is reached after a brief period of search. The agent then moves towards the feeder which again it narrowly misses and overshoots. After another brief search the feeder is found. The rest of the trial shown consists of orderly movement between nest and feeder. Other brief periods of search occur later on in the trial (not shown).

SV and HV were reset frequently enough. An interesting feature of the system’s behaviour was noted, namely that the dynamics of homing and search tend to produce an initial overshoot when reaching the expected goal location which might conceivably act to compensate for HV decay-induced undershoot, especially in the case where the agent spends most of its time moving in straight lines between nest and feeder.

Little can be said about the chances of successfully evolving a vector navigation system on the basis of the hand built network shown here. Clearly CiTRuS can implement such a system, using a relatively small number of components.
The main results of a modelling study focused on PI must of course relate to this subject. Since we have deployed a number of new modelling tools we are also in a position to comment on the performance of these but naturally we cannot draw conclusions which go much beyond the scope of the PI tasks for which they were utilised.

8.1 Conclusions Relating to Path Integration

8.1.1 The One Dimensional Tasks

Here we saw two ways that a leaky integrator CTRNN neuron could be tuned to act as a reasonably accurate integrator (see Sections 4.3.1 and 4.3.2). Both mechanisms involved counteracting the decay term of the leaky integrator to make it more closely approximate a non-leaky integrator. The two-way task agent had the output of the integrator neuron modify the value of the forward motor neuron which then fed back into the integrator neuron. The one-way task had a direct self weight on the integrator neuron performing the same role.

It may seem strange that so much of the analysis (including of the two dimensional PI agents) has focused on the use of leaky integrators along with mechanisms to neutralise the effects of leakage. It would seem more logical at first glance to have allowed accurate integrators, with no decay processes imposed on them. There are at least two reasons for resisting this temptation. Firstly for practical reasons, when employing evolvable controllers of any kind, it is more convenient to use a dynamical system which can easily be prevented from displaying divergent dynamics. One way to achieve this is to sigmoid the output of all neurons, and make them act as leaky integrators. Sigmoidal outputs prevent exponential growth in neurons with positive-feedback self connections, a leakage term further limits the neuron state to remain within the same range as the neuron’s input. This helps prevent numerical variables in the simulation code growing very large, and helps prevent artefactual behaviour in a numerical integration engine. Secondly biological nervous systems are themselves finite and clearly cannot instantiate a perfect integrator in the mathematical sense, since this would imply they could attain arbitrarily large values for some physical property or other simply by integrating a fixed input for a long period of time. Therefore we might well expect a biological system to have to employ roughly analogous tricks to those found in the networks...
presented here, to overcome the limitations of their nervous systems and approximate accurate integration, at least over a limited range of conditions. Whether actual single neuron integrators (such as Egorov et al., 2002, found) perform this kind of role in nature is not currently known for PI.

8.1.2 PI in Two Dimensions

Mathematical Results

Having noted the four canonical coordinate systems applicable to PI in two dimensions, it was noted that these are four alternative ways of describing the same process. This enabled the translation of the geocentric Cartesian Mittelstaedt bicomponent model (in its simplest form, and lacking any forward speed control) into an egocentric polar form which enabled a simple analysis of the dynamics of homing, including the proof that this version of the system is reversible. The conclusion reached for this model was that the trajectories should not be interpreted as search patterns, but only as homing to the vicinity of the nest, and that the turning rate constant should be sufficiently high to allow the animal to return close to the nest.

It was noted that the addition of a simple leakage process to the HV update equations can reproduce the systematic homing errors reported in *C. fortis* ants in Müller and Wehner (1988). It was further noted that this leakage process can be readily translated into all of the canonical coordinate systems with resulting identical abilities to account for the ant’s behaviour. The four coordinate systems therefore remained alternative descriptions of the same process.

To explain the accurate homing ability of the evolved PI agent despite the small time constant of its HV integrators, a system was introduced whereby the HV was stored on leaky integrators, but HV input was progressively scaled down over time to match the leakage process. This method of leakage correction was found to enable accurate navigation over a limited time scale. It was also found to be translatable into all four coordinate systems and usable for a vector navigation system whereby the HV was stored in higher order memory for later retrieval.

Given the possibility of similar or identical behaviour of PI systems based on all four coordinate systems, even with the presence of uncorrected leaky integration errors, it may not be possible to decide from behavioural data alone how a PI system in nature works internally, even assuming it uses a straightforward implementation of one of them. Further work studying the effects of errors and perturbations introduced to the ‘basic four’ models might suggest the best behavioural tasks to set the animals in order to yield the most sensitive test of the internal PI mechanism. Manipulations of the sensory input of *Cataglyphis* have already been undertaken including both compass (Fent, 1986) and potential odometric (Ronacher and Wehner, 1995; Ronacher et al., 2000; Wittlinger et al., 2006) cues, and might also prove very useful in this regard. It may at least be possible to devise tests to distinguish between a leaky integration model and the Müller-Wehner model.

Simulation Results

For the first time a neural model of PI was produced using artificial evolution rather than human design or supervised learning. The only requirement for the agent’s behaviour was that it returned to the nest as rapidly as possible. It was constrained to use a speed sensor and two sinusoidal response function allothetic compass sensors. No constraints, other than maximum size, were placed on the internal topology of the control network.
Having avoided constraining the internal functioning of the network, the next major constraint on the resulting PI system must be the sensory and motor morphology available to the agent. For the evolved-network allothetic compass experiments, the shape of the compass response function was varied to determine what influence this had on the resulting controllers. The best results were obtained when the function was a pure cosine. This is the function most easily used to produce a HV since, with an angle of $\frac{\pi}{2}$ radians between the orientation of the two sensors on the agent’s body, this is equivalent to having the sine and cosine functions required for a geocentric Cartesian HV. When a linearised version of cosine was used, the network evolved to treat them as if they were pure cosine sensors, and did not therefore make any correction for the resulting errors. Sensors further removed from cosines gave even worse results. This suggests that the methods used were unable to construct networks using compass response functions of arbitrary shape, and that accurate PI is harder or perhaps impossible to artificially evolve if the most suitable sensors are not available. This contrasts with the results of Wittmann and Schwegler (1995) who present a network model of PI able to use any single peaked compass response function.

The agent evolved to solve the PI task by using an internal estimate of its location which it updated continuously as it moved around the arena, i.e. a HV. This point may seem obvious, but it is worth stating that there was no formal constraint on the network to function in this manner. Of course it is the best and simplest way to solve the task. Alternatives include: remembering the complete structure of the outward journey until homing is required; constraining the motion of the agent to radial movements from the nest (i.e. the agent would have to return to the nest after visiting each beacon), thus requiring only the distance from the nest to be memorised during each radial movement; performing an exhaustive search for the nest.

The evolved agent used a non-conventional HV, in that the values decayed significantly over the duration of the agent’s journey, but without the resulting homing behaviour being correspondingly inaccurate. This was achieved using a leakage correction mechanism. Leakage correction, whilst mathematically understandable as a method of perfectly correcting for HV decay, was implemented in such a way that correction was accurate during the initial part of the agent’s journey only. This was understandable as a way of preventing the agent's rate of turn from decaying to zero along with the decaying HV. The resulting system was a compromise between perfect leakage correction and high turning rates, such that correction was accurate when the agent was away from the nest, but inaccurate whilst it was searching near the nest when errors were likely to be less important.

The agent evolved to return home using a straight, direct path, and to search in the vicinity of the nest using a pattern which had a radially symmetrical bell shaped search density and which did not fall into a repeating pattern even in the absence of sensory and motor noise. The method of generating the search pattern is unlike any existing model of search in desert ants (Wehner and Srinivasan, 1981; Müller and Wehner, 1994), in that the PI-mediated homing system and the search generation system are combined into a single unitary mechanism. The rotation of the agent was found to be homologous to a linearly-damped pendulum, where the strength of gravity was proportional to the distance from the HV origin. This generated straight, homeward trajectories when the agent was far from the nest and twisting, multidirectional movement when close to the nest. A brief survey of the parameter space of the homing and search system showed that the
trajectories were complex and variable, and the efficiency of the trajectory as a search pattern varied considerably with the parameter values. This suggests that hand designing such a search system may have been difficult. The search patterns of the evolved agent have not been proven to be optimal in any way, and further work would be useful in this respect.

Both the search generation and leakage correction mechanisms were found to be equally applicable to a PI system using an egocentric Cartesian HV, and to a vector navigation system which stored a geocentric Cartesian HV in a higher order memory for later retrieval. Thus both of the novel features evolved in the earlier PI experiment were shown to be applicable beyond the narrow scope of that simulation. Future work might attempt to evolve a vector navigation system or a PI system using an idiothetic compass, or alternatively the hand designed systems presented here could be tuned for efficiency using evolution.

As stated in the introduction, evolutionary robotics and the dynamical approach to cognition emphasises the importance of the coupling which exist between an agent’s brain, body and environment. Here the explicit simulation of the agent’s trajectory, rather than a model dealing simply with the state of the HV, or one which does not simulate the homing part of the journey, allowed the generation of a combined homing and search model. Future work could model the sensory and motor morphology of desert ants more closely than was attempted here, such that a more detailed comparison of behaviours could be undertaken. A more realistic model of the foraging behaviour would also be possible. The features of the ant’s environment necessary for PI are relatively simple to model, including potentially the pattern of skylight polarisation.

8.2 Conclusions Relating to Neural Networks

We have not employed neural networks as biologically realistic models of neurons in this work. Rather we have treated the neural network controller of the agent as a more abstract entity, a dynamical system. The CTRNN model can clearly be derived from a (greatly simplified) description of the behaviour of the membrane potential of biological neurons (see for example Koch, 1999; Dayan and Abbott, 2001), but is also a convenient and relatively well studied dynamical system for the purposes of simulation modelling with genetic algorithms. The proof that CTRNN is capable of approximating any smooth dynamical system given sufficient neurons (Funahashi and Nakamura, 1993), the efforts being made to understand its general dynamical properties (Beer, 1995, 2006) and its continued successful use as an evolvable agent and robot controller (Gallagher and Beer, 1999; Macinnes and Di Paolo, 2004; Goldenberg et al., 2004; Izquierdo-Torres and Harvey, 2006, for example) make it a good choice as the starting point for any exploration of dynamical agent controllers.

8.2.1 Comparison of CTRNN to ModCTRNN

The allothetic-compass PI experiments provided a direct comparison between CTRNN and ModCTRNN, using the same tasks and the same GA. A successful CTRNN controller was only found for the simplified task where the agent moved at a constant speed, and thus did not need to take account of the speed sensor output to perform PI. Here the best CTRNN agent was able to reach home very reliably, but was unable to travel there by a direct, straight path. It thus underperformed relative to the ModCTRNN on this version of the task. The Mittelstaedt model suggests a reason
for this, namely that multiplication of the compass sensor outputs by the contralateral HV component is the simplest way to generate straight homing behaviour. CTRNN cannot multiply two variables together directly, and the GA was apparently unable to build a multiplier incrementally from multiple CTRNN nodes in the time available to it. ModCTRNN can, in a slightly roundabout way, implement direct multiplication of two variables. One of the variables must be a weight (or be copied to a weight) with a low time constant. This weight can then be used to multiply the other variable, which must be a sensor or node output, and input the product into one (and only one) receiving node. Given this less than ideal way of directly performing multiplication the ModCTRNN was able to outperform the CTRNN by homing in a straight line.

For the variable speed PI task the CTRNN did not evolve any kind of solution, suggesting it was unable to evolve to correctly update a HV, even approximately. The simplest HV update equations for the compass sensors available clearly require a multiplication, this time of the speed sensor by a compass sensor. It therefore appears that the same basic limitation of the CTRNN is responsible for its failure here. ModCTRNN evolved to not only multiply the speed and compass sensor values, but also to multiply this product by a third value, the leakage correction value. This was achieved by chaining weights together and shows how the apparent limitation of only being able to multiply two values together can be circumvented.

The CTRNN solution to the one-way one-dimensional task - negative feedback of speed towards a constant value - is not viable on its own due to the way speed perturbations were introduced in the 2D task. The agent was initially allowed to move at a constant speed but was held stationary at the last beacon before being allowed to home. Only if this task was solved were further random perturbations applied to forward speed during the whole journey.

The ModCTRNN shows the potential of an evolvable controller capable of directly multiplying values together. PI provides an example of a task where this operation is clearly very useful, although it may not be strictly essential (Conklin and Eliasmith (2005) provide a spiking neuron model of PI in rodents, based on the known properties of place cells, which does not require multiplicative synapses). The resulting neural network was small, efficient and readily analysable.

### 8.2.2 CiTRuS

CiTRuS implements multiplication directly using multiplier nodes, such that any number of variables can be multiplied together. In this way it is a more logical and direct implementation of multiplication. It is also designed to prevent unlimited increase in any network variable, by applying a sigmoid function to the output of all nodes, a feature which ModCTRNN weights lacked, allowing them to increase without limit (but only at a constant rate) under certain conditions. The CiTRuS latching feature, whilst not tested in any evolutionary experiments, was shown to be a powerful enhancement over ModCTRNN, making the storage and later retrieval of the HV possible in a straightforward way.

Another possible advantage of CiTRuS over ModCTRNN is that it is possible to define a fully connected network topology, given only the number of ordinary and multiplier nodes present. ModCTRNN has the property of creating a new node with every weight added, and so can never become fully connected. This was felt to be important since it would allow a fixed length encoding scheme with CiTRuS, which is a much simpler kind of genotype to deal with than a variable length
one. This feature would allow CiTRuS to be tested against CTRNN using the same methods as Beer has employed in his extensive series of agent experiments.

Future work is certainly required to test CiTRuS as an evolvable dynamical system controller, and to see whether its novel features will allow it to outperform CTRNN and ModCTRNN on certain tasks, and whether it can indeed yield simpler, more easily analysible controllers.

8.3 Conclusions Relating to Genetic Algorithms

8.3.1 Automatic Network Pruning

We have explored the use of a generalised variable length encoding scheme and an associated set of mutation operators, referred to as GeNGA. GeNGA allowed the use of automated stochastic pruning of the evolved ModCTRNN and CTRNN networks. The pruned ModCTRNN was shown to contain no redundant features, and retained a fitness level at least approximately the same as the unpruned network, indicating that pruning was successful. This simplified the subsequent analysis of the network since, unlike the unpruned one dimensional task agents, analysis did not have to begin with the ad hoc identification and removal of suspected redundant network components. This is an advantage of using a variable length encoding over a fixed length one.

8.3.2 Generality Versus Specialisation

GeNGA was equally capable of encoding and mutating ModCTRNN and CTRNN networks (and could also have dealt with CiTRuS networks). The requirement for the generality of the mutation operators necessarily made advanced features such as metamutational ratchet deletions more complex to implement, and so only a simple set of operators was used. The interesting question of whether the use of a set of operators specially designed to create and exploit neutral networks in genotype space has thus not been addressed. This would require a comparison between GAs using the same network model on the same task, with different sets of mutation operators.

8.4 Conclusion

To summarise, the results presented suggest that, in evolutionary robotics using a more complex, variable length encoding scheme and a more complex artificial neural network can lead, at least sometimes, to a smaller, more efficient and more easily analysible dynamical system model of the behaviour being studied.

Regarding PI, the review, simulation results and analysis suggest modellers should not forget the equivalence of the four main mathematical formulations. Previous work has mainly focused on advocating one formulation over the others. The simulation results show that features evolved using one representation can sometimes be readily translated into another (as shown by the leakage correction and searching mechanisms). Analysis can be made more convenient, as illustrated during the analysis of the evolved PI system, where twice an egocentric polar formulation yielded a concise result (reversibility and homology to a pendulum) regarding an apparently exclusively geocentric Cartesian PI system. Leaky HVs were shown to give rise to identical homing errors in every coordinate system.
Whilst the results presented here do not strengthen the position of the Müller-Wehner model, since they provide an alternative explanation for the observed errors (namely leaky integration), at a higher level there is agreement on at least one point. The Müller-Wehner model aimed to show that it was not necessary to postulate an implementation of geometrically accurate PI to explain the ants’ homing ability, since a possibly simpler ‘formalism’ existed sufficient for the ants’ needs. Here we can agree that any process capable of bringing the ant home could be used to explain its behaviour, and that this need not be a straightforward conversion of geometrical equations into neural dynamics. We have generated a PI system which can operate with high precision regardless of the continuous decay of stored spatial information over time, and hence where the agent’s subjective world is perpetually shrinking.
Bibliography


Bibliography


ten*, 87:133–136.


Bibliography


Appendix A

Glossary and Abbreviations

- 1D - one dimensional
- 2D - two dimensional
- agent - simulated animal
- allothetic - externally generated
- Cartesian - (in this thesis) a system of rectilinear two dimensional coordinates \((x, y)\), where the axes are perpendicular, and the scale factors are both unity
- CiTRuS - continuous time recurrent system
- CTRNN - continuous time recurrent neural network
- EC - egocentric Cartesian
- egocentric - self centred
- EP - egocentric polar
- exocentric - externally centred, defined relative to the external environment
- GA - genetic algorithm
- GC - geocentric Cartesian
- geocentric - defined relative to the external environment
- GP - geocentric polar
- HV - home vector
- idiothetic - self generated
- L. - Linnaeus (denotes the species was classified and named by Carl Linnaeus)
- ModCTRNN - modified CTRNN
- Ornstein-Uhlenbeck Process - continuous time stochastic process which, unlike a random walk, has the tendency to return towards a certain mean value
- PI - path integration
- polar - polar coordinates \((r, \gamma)\) are defined relative to Cartesian coordinates \((x, y)\) as \(x = r \cos \gamma, y = r \sin \gamma\)
- rectangular - rectangular coordinates, synonym for Cartesian coordinates
Appendix B

Mathematical Glossary

This appendix summarises some mathematical results and terminology necessary to follow the mathematical derivations in this thesis.

B.1 Logarithms

\[ \ln \frac{a}{b} = \ln a - \ln b \]

B.2 Differentiation

Shorthand for first order derivative used later in this section only (but note I have used prime notation elsewhere to denote egocentric coordinates):

\[ y = f(x) \]
\[ \frac{dy}{dx} = y' \]

Shorthand for first order derivative with respect to time:

\[ y = f(t) \]
\[ \frac{dy}{dt} = \dot{y} \]

The chain rule:

\[ \frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx} \]

The product rule:

\[ x = uv \]
\[ x' = u'v + uv' \]

The quotient rule:

\[ x = \frac{u}{v} \]
Sine and cosine:

\[
\begin{align*}
\sin(ax) &= a\cos(ax) \\
\cos(ax) &= -a\sin(ax)
\end{align*}
\]

**B.3 Integration**

\[\alpha\dot{x} = -x + k\]

where \(\alpha\) and \(k\) are constants, integrates to:

\[x = (x_0 - k)e^{-\frac{x}{\alpha}} + k\]

**B.4 Trigonometry**

\[
\tan x = \frac{\sin x}{\cos x}
\]

\[
\begin{align*}
\cos(-x) &= \cos x \\
\sin(-x) &= -\sin x \\
\cos(x + \pi) &= -\cos x \\
\sin(x + \pi) &= -\sin x \\
\cos\left(x \pm \frac{\pi}{2}\right) &= \mp \sin x \\
\sin\left(x \pm \frac{\pi}{2}\right) &= \pm \cos x \\
\sin^2 x + \cos^2 x &= 1 \\
\cos(x + y) &= \cos x \cos y - \sin x \sin y \\
\sin(x + y) &= \sin x \cos y + \cos x \sin y
\end{align*}
\]

Rotation of \((x, y)\) anticlockwise by \(\theta\) radians about the origin gives:

\[
(x \cos \theta - y \sin \theta, x \sin \theta + y \cos \theta)
\]

**B.5 Miscellaneous**

\(X \sim N(\mu, \sigma^2)\) - \(X\) is a random value drawn from the normal distribution with a mean of \(\mu\) and a variance of \(\sigma^2\).
Appendix C

Reversibility of the Embedded Two Variable Egocentric Polar System within the Mittelstaedt Model

The following shows that the two variable system, \((r', \gamma')\), embedded in the Mittelstaedt PI system is reversible. The linearisation of the system classifies the two fixed points as neutrally stable centres (see Section 2.3.5). If the full, non-linear system is shown to be reversible then it follows that its fixed points are also neutrally stable centres (Strogatz, 1994).

The two variable embedded system:

\[
\begin{align*}
\dot{r}' &= -\cos \gamma' \\
\dot{\gamma}' &= \left(\frac{1}{r'} - kr'\right) \sin \gamma'
\end{align*}
\]

whose fixed points are at \((\sqrt{\frac{1}{k}}, \frac{\pi}{2})\) and \((\sqrt{\frac{1}{k}}, -\frac{\pi}{2})\). We begin by defining systems where the fixed points are at the origin. For the first fixed point we have:

\[
\begin{pmatrix}
a_1 \\
b_1
\end{pmatrix} = \begin{pmatrix}
r' - \sqrt{\frac{1}{k}} \\
\gamma - \frac{\pi}{2}
\end{pmatrix}
\]

and for the second:

\[
\begin{pmatrix}
a_2 \\
b_2
\end{pmatrix} = \begin{pmatrix}
r' - \sqrt{\frac{1}{k}} \\
\gamma + \frac{\pi}{2}
\end{pmatrix}
\]

giving:

\[
\begin{pmatrix}
a_1 \\
b_1
\end{pmatrix} = \begin{pmatrix}
a_2 \\
b_2
\end{pmatrix} = \begin{pmatrix}
r' \\
\dot{\gamma}'
\end{pmatrix}
\]

For the \((a_1, b_1)\) system:

\[
a_1 = -\cos \gamma' = -\cos(b_1 + \frac{\pi}{2}) = \sin b_1
\]
\( \dot{b}_1 = \left( \frac{1}{r'} - kr' \right) \sin \gamma = f(a_1) \sin (b_1 + \frac{\pi}{2}) = f(a_1) \cos b_1 \)

and for the \((a_2, b_2)\) system:

\( \dot{a}_2 = -\cos \gamma' = -\cos(b_2 - \frac{\pi}{2}) = -\sin b_2 \)

\( \dot{b}_2 = \left( \frac{1}{r'} - kr' \right) \sin \gamma' = f(a_2) \sin (b_2 - \frac{\pi}{2}) = -f(a_2) \cos b_2 \)

(the function \( f(z) = \frac{1}{z + \sqrt{\frac{1}{k}}} - k(z + \sqrt{\frac{1}{k}}) \) turns out to be unimportant). The system is reversible if \( \dot{a}_1(a_1, -b_1) = -\dot{a}_1(a_1, b_1) \) and \( \dot{b}_1(a_1, -b_1) = \dot{b}_1(a_1, b_1) \). This is true since \( \sin(-b_1) = -\sin b_1 \) and \( \cos(-b_1) = \cos b_1 \). Likewise for \((a_2, b_2)\). This proves that all trajectories “sufficiently close” to the fixed points of \((r', \gamma')\) are closed (Strogatz, 1994).
Appendix D

Leaky integration model for navigation errors after long straight journeys

We use the leaky PI model defined by Eqn. 2.11 to predict the homing distances of ants after they have performed long straight outward journeys to a feeder before beginning to home. The model has a single parameter, the time constant of the leaky integrators \( \alpha \) which can be used to fit the model to experimental data.

It is simplest to proceed from the egocentric polar description of the leaky PI system (Eqns. F.6):

\[
\begin{align*}
\dot{z}' &= -\frac{z'}{\alpha} - s \cos \zeta' \\
\dot{\zeta}' &= \frac{1}{z'} s \sin \zeta' - \dot{\theta}
\end{align*}
\]

We assume the ant’s speed is constant throughout the journey and that it does not stop or search at the feeder for any significant length of time. Since the time units are arbitrary, we further assume speed to be unity. On the outward trip the ant heads directly away from the nest, on the return trip it heads directly towards it, hence the \( \cos \zeta' \) term in the \( \dot{z}' \) equation will be \(-1\) and \(1\) for the outward and return legs respectively. Note that \( \dot{\zeta}' = 0 \) whenever the ant is heading towards or away from estimated home, so that \( \zeta' \) will not vary during either leg of the journey, but will change abruptly from \( \pi \) to \(0\) after reaching the feeder as the ant begins homing. The equation for \( \dot{z}' \) during the outward and return legs will, respectively, be:

\[
\begin{align*}
\alpha \dot{z}' &= -\dot{z}' + \alpha \\
\alpha \dot{\zeta}' &= -\dot{z}' - \alpha
\end{align*}
\]

Walking at a unitary speed it takes \( x \) time units to walk a distance \( x \), giving a \( z' \) value at the feeder of (see Appendix B):

\[
z'_f = \alpha \left(1 - e^{-\frac{x}{\alpha}}\right)
\]

assuming \( z' \) is initialised to \(0\). For the return journey we have this as the initial value of \( z' \), and require the time taken, \( y \), for it to reduce to zero:
\[(z' + \alpha) e^{-y} - \alpha = 0\]

implies that:

\[y = \alpha \log \left( 2 - e^{-\frac{z}{\alpha}} \right)\]
Appendix E

Derivation of leakage corrected PI models for all four coordinate systems

The evolved PI system presented in chapter 5 used a mechanism which allowed accurate navigation despite using leaky integrators to store the HV. This appendix shows how navigation can remain accurate while the HV is constantly decaying for the geocentric Cartesian case. Equivalent models are obtained for all three remaining coordinate systems which likewise allow accurate navigation with decaying HVs.

E.1 Leakage Corrected Geocentric Cartesian

Below \((x,y)\) is the true geocentric Cartesian coordinate of the agent, \((u,v)\) is the agent’s leaky geocentric Cartesian HV (which would be stored on weight pair two in the evolved network, Fig. 5.8), \(I_u = s \cos \theta, I_v = s \sin \theta, s\) the agent’s forward speed and \(\theta\) it’s compass heading \((I_u \text{ and } I_v\) are equivalent to the product of the speed sensor and either of the compass sensors in the evolved network, Fig. 5.8):

\[
\dot{x} = I_u \\
\dot{y} = I_v
\]

We know the HV is stored on two ModCTRNN weights, which have exponential decay behaviour. We therefore guess the relationship between \((x,y)\) and \((u,v)\), where \(k\) is a constant scaling factor and \(\alpha\) is the decay time constant:

\[
u = ke^{-\frac{t}{\alpha}} \int I_u dt
\]

Which, using the relation \(u = ke^{-\frac{t}{\alpha}} \int I_u dt\), simplifies to:
\[ \alpha \dot{u} = -u + \alpha ke^{-\frac{t}{\alpha}} I_u \]

Implying that for the relation \( u = kxe^{-\frac{t}{\alpha}} \) to hold the input to the integrator must be scaled by an exponentially decaying term \( \alpha ke^{-\frac{t}{\alpha}} \). This can be provided by a ModCTRNN weight \( w \):

\[ \alpha \dot{u} = -u + ws \cos \theta \]  \hspace{1cm} (E.2)

with \( \alpha \dot{w} = -w \) and \( w_0 = \alpha k \). Likewise for \( v \) (which uses a weight with the same decay behaviour as \( u \)):

\[ \alpha \dot{v} = -v + ws \sin \theta \]  \hspace{1cm} (E.3)

Since the homing equation relies only on the ratio of the two HV components, the decayed HV can still allow accurate homing to take place as long as the weight has not decayed to zero:

\[ \dot{\theta} = u \sin \theta - v \cos \theta = ke^{-\frac{t}{\alpha}} (x \sin \theta - y \cos \theta) \]  \hspace{1cm} (E.4)

To summarise, we have shown that the relationship E.1 holds provided E.2 and E.3 are observed, and that this is sufficient to allow accurate homing to occur using E.4, subject to the fact that the effective turning rate coefficient decays exponentially over a time scale determined by \( \alpha \).

### E.2 Leakage Corrected Geocentric Polar

We now wish to define the equivalent PI model in geocentric polar. Below \((r, \gamma)\) is the true geocentric polar coordinate of the agent and \((z, \zeta)\) is the leaky geocentric polar HV. First we define \( z = \sqrt{u^2 + v^2} \), which, using the relation \( r = \sqrt{x^2 + y^2} \), implies:

\[ z = ke^{-\frac{t}{\alpha}}r \]  \hspace{1cm} (E.5)

We define \( \zeta = \arctan \frac{v}{u} \). Using the relationship given by E.1 this gives \( \zeta = \arctan \frac{\zeta}{\dot{\zeta}} \) and therefore:

\[ \zeta = \gamma \]

Defining \( I_z = s \cos(\theta - \zeta) \) and noting \( \dot{r} = I_z \), we have the same form as the relation between \( u \) and \( x \), therefore we have (with \( \alpha \dot{v} = -w \) and \( w_0 = \alpha k \)):

\[ \alpha \dot{z} = -z + ws \cos(\theta - \zeta) \]

Since \( \dot{\zeta} = \dot{\gamma} \) and using the relation E.5 we have:

\[ \alpha \dot{\zeta} = \frac{1}{z} ws \sin(\theta - \zeta) \]

### E.3 Leakage Corrected Egocentric Cartesian

Next we derive a leakage corrected egocentric Cartesian PI system, using the same methods as for the geocentric Cartesian case. Below \((x', y')\) is the true egocentric Cartesian coordinate of the nest
relative to the agent’s position, \((u', v')\) is the leakage corrected egocentric Cartesian HV. We start with the accurate HV update equations:

\[
\begin{align*}
\dot{x}' &= \hat{\theta}y' - s \\
\dot{y}' &= -\hat{x}'
\end{align*}
\]

Defining the same relation as we did between \((x, y)\) and \((u, v)\):

\[
\begin{align*}
u' &= kx'e^{-\frac{t}{\alpha}} \\
v' &= ky'e^{-\frac{t}{\alpha}}
\end{align*}
\]  \hspace{1cm} (E.6)

we find \(\dot{u}'\) using the quotient rule:

\[
\dot{u}' = ke^{-\frac{t}{\alpha}} \left( \frac{\dot{x}'}{\alpha} - \frac{x'}{\alpha} \right)
\]

Substituting for \(x'\) and \(\dot{x}'\) we find:

\[
\alpha \dot{u}' = -u' + \alpha \hat{\theta}v' - \alpha ke^{-\frac{t}{\alpha}}s
\]

which can be implemented once again using a decaying weight \(w\):

\[
\alpha \dot{u}' = -u' + \alpha \hat{\theta}v' - ws
\]  \hspace{1cm} (E.7)

where \(\alpha \dot{w} = -w\) and \(w_0 = \alpha k\) as before. Applying the same process to \(v'\) yields:

\[
\alpha \dot{v}' = -v' - \alpha \hat{\theta}u'
\]  \hspace{1cm} (E.8)

The non-leaky homing equation is (with the turning rate parameter omitted):

\[
\dot{\theta} = v'
\]

The leakage corrected version would therefore be:

\[
\dot{\theta} = v' = ke^{-\frac{t}{\alpha}}y'
\]

and, as with the geocentric case, we have a system which still homes accurately, subject to an exponentially decaying turn rate coefficient controlled by \(\alpha\).

### E.4 Leakage Corrected Egocentric Polar

Finally we convert the egocentric Cartesian model into polar form in the same way as we did for the geocentric models. \((r', \gamma')\) is the true egocentric polar coordinate of the nest, \((\zeta', \zeta')\) the leakage corrected HV. We use the relationships \(\zeta' = \sqrt{u'^2 + v'^2}\) and \(\zeta' = \arctan \frac{v'}{r'}\) giving (in the same manner as the geocentric leakage corrected case):

\[
\begin{align*}
z' &= ke^{-\frac{t}{\alpha}}r' \\
\zeta' &= \gamma'
\end{align*}
\]
Defining \( I_z = s \cos \zeta' \) and noting \( \dot{r}' = I_z \) we have the same relationship as for the geocentric case, giving:

\[
\begin{align*}
\alpha' &= -z' - ws \cos \zeta' \\
\dot{\alpha}' &= \frac{1}{2} ws \sin \zeta' - \alpha \theta
\end{align*}
\]

**E.5 Conclusion**

It should be emphasised that all four leakage corrected models are alternative ways to describe the same system; the agent’s homing behaviour would be identical provided noise were absent. This could be shown more conclusively by deriving the egocentric Cartesian case by mapping from the geocentric Cartesian using the relationship \((x', y') = (-x \cos \theta - y \sin \theta, x \sin \theta - y \cos \theta)\), rather than the method used above. Appendix F shows equivalence in this direct way for the more general case where HV integrators are leaky, but the leakage correction mechanism may or may not be active, and hence contains the system dealt with in this chapter as a special case.
Appendix F

Proof of equivalence of the four leaky PI models

The following shows that the leaky PI models in all four coordinate systems produce identical behaviour. The proof (outlined below) applies whether the leakage correction weight decays with time constant $\alpha$ or remains constant at its initial value. Hence the result is that all four models give exactly the same homing behaviour whether or not HV leakage is compensated for. This means that the systematic errors generated by uncorrected leakage are identical across all four coordinate systems.

We begin with leaky PI equations in geocentric polar and Cartesian, then show that the relation $(x, y) = (r \cos \gamma, r \sin \gamma)$ is always true. Likewise for egocentric leaky PI, $(x', y') = (r' \cos \gamma', r' \sin \gamma').$ Finally it is shown that the relation $(x', y') = (-x \cos \theta - y \sin \theta, x \sin \theta - y \cos \theta)$ is true.

F.1 Geocentric Equations

Using the symbols from Appendix E ($(x, y)$ true GC coordinate, $(u, v)$ equivalent leaky HV, $(r, \gamma)$ true GP coordinate, $(z, \zeta)$ equivalent leaky HV):

\[
\begin{align*}
\alpha \dot{u} &= -u + \omega s \cos \theta \\
\alpha \dot{v} &= -v + \omega s \sin \theta \\
\alpha \dot{z} &= -z + \omega s \cos(\theta - \zeta) \\
\alpha \dot{\zeta} &= \frac{1}{\varsigma} \omega s \sin(\theta - \zeta)
\end{align*}
\] (F.1)

We wish to show that $u = z \cos \zeta$. It can be shown that the derivative of $z \cos \zeta$ is equal to that defined for $u$. Using the chain and product rules we differentiate $u = z \cos \zeta$:

\[
\dot{u} = \dot{z} \cos \zeta - z \dot{\zeta} \sin \zeta
\]

Substituting for $\dot{z}$ and $\dot{\zeta}$ using F.2:

\[
\alpha \dot{u} = \cos \zeta [\omega s \cos(\theta - \zeta) - z] - \sin \zeta [\omega s \sin(\theta - \zeta)]
\]

Using the formulae for $\cos(x + y)$ and $\sin(x + y)$ (see Appendix B) and the relation $z = \frac{\varsigma}{\cos \zeta}$.
\[ \alpha \dot{u} = \cos \zeta \left[ w_s \cos \theta \cos \zeta + w_s \sin \theta \sin \zeta - \frac{u}{\cos \zeta} \right] - \sin \zeta \left[ w_s \sin \theta \cos \zeta - w_s \cos \theta \sin \zeta \right] \]

Using the identity \( \cos^2 x + \sin^2 x = 1 \) this simplifies to:

\[ \alpha \dot{u} = -u + w_s \cos \theta \]

Similarly using the relation \( v = z \sin \zeta \) it can be shown that the derivative of \( z \sin \zeta \) is equal to that defined for \( v \). Since these relations do not depend on the behaviour of \( w \) over time, we can define \( w \) to perform leakage correction (\( \alpha \dot{w} = -\dot{w}, w_0 = \alpha k \)) or to remain constant (\( \dot{w} = 0 \)).

### F.2 Egocentric Equations

We proceed in exactly the same way as for the geocentric equations. Using the symbols from Appendix E ((\( x', y' \)) true EC coordinate, (\( u', v' \)) equivalent leaky HV, (\( r', \gamma' \)) true EP coordinate, (\( z', \zeta' \)) equivalent leaky HV):

\[ \begin{align*}
\alpha \dot{u}' &= -u' + \alpha \dot{\theta} v' - w_s \\
\alpha \dot{v}' &= -v' - \alpha \dot{\theta} u' \\
\alpha \dot{z}' &= -z - w_s \cos \zeta' \\
\alpha \dot{\zeta}' &= \frac{1}{z} w_s \sin \zeta' - \alpha \dot{\theta}
\end{align*} \]

We wish to show that \( u' = z' \cos \zeta' \) as before. Likewise it can be shown that the derivative of \( z' \cos \zeta' \) is equal to that defined for \( u' \). Using the chain and product rules we differentiate \( u' = z' \cos \zeta' \):

\[ \dot{u}' = -z' \dot{\zeta}' \sin \zeta' + z' \cos \zeta' \]

Substituting for \( \dot{z}' \) and \( \dot{\zeta}' \) using F.4 we obtain:

\[ \alpha \dot{u}' = -z' \sin \zeta' \left[ w_s \sin \zeta' - \alpha \dot{\theta} \right] + \cos \zeta' \left[ -z' - w_s \cos \zeta' \right] \]

Using \( \cos^2 x + \sin^2 x = 1 \), \( u' = z' \cos \zeta' \) and \( v' = z' \sin \zeta' \) we simplify and obtain the equation for \( \dot{u}' \) given in F.3. We proceed similarly to differentiate the relation \( v' = z' \sin \zeta' \):

\[ \dot{v}' = -z' \dot{\zeta}' \cos \zeta' + z' \sin \zeta' \]

Substituting for \( \dot{z}' \) and \( \dot{\zeta}' \) using F.4 we obtain:

\[ \alpha \dot{v}' = \dot{z}' \cos \zeta' \left[ w_s \sin \zeta' - \alpha \dot{\theta} \right] + \sin \zeta' \left[ -z' - w_s \cos \zeta' \right] \]

which simplifies to the equation for \( \dot{v}' \) given in F.3.

### F.3 Linking the geo- and egocentric leaky models

The relationship between the true GC and EC coordinates is given by:
\[ x' = -x \cos \theta - y \sin \theta \]
\[ y' = x \sin \theta - y \cos \theta \]

We wish to show that this relation holds for the leaky models, namely:

\[ u' = -u \cos \theta - v \sin \theta \]
\[ v' = u \sin \theta - v \cos \theta \]  \hspace{1cm} (F.5)

Using the chain and product rules we differentiate \( u' = -u \cos \theta - v \sin \theta \):

\[ \dot{u}' = u \dot{\theta} \sin \theta - \dot{u} \cos \theta - v \dot{\theta} \cos \theta - \dot{v} \sin \theta \]

Substituting for \( \dot{u} \) and \( \dot{v} \) using F.1, and using the relations given in F.5, this simplifies to the equation for \( \dot{u}' \) given in F.3. Using the same approach we can differentiate \( v' = u \sin \theta - v \cos \theta \):

\[ \dot{v}' = u \dot{\theta} \cos \theta + \dot{u} \sin \theta + v \dot{\theta} \sin \theta - \dot{v} \cos \theta \]

Substituting for \( \dot{u} \) and \( \dot{v} \) using F.1, and using the relations given in F.5, this simplifies to the equation for \( \dot{v}' \) given in F.3.

### F.4 Conclusions

Unlike Appendix E, where we had to set \( \alpha \dot{w} = -w \) and \( w_0 = \alpha k \) in order to ensure accurate homing, here we placed no restrictions on \( w \), but still obtained the result that all four models give exactly equivalent behaviour. This implies that the systematic homing errors introduced by deviating from the leakage correction dynamics of \( w \) (such as by keeping \( w \) constant at 1) are the same for all four models given an equivalent homing system.

Simplified forms of the leaky equations, which do not contain the leakage correction mechanism are as follows (obtained by setting \( k = 1 \), \( \dot{w} = 0 \) and \( w_0 = \alpha \)):

\[ \dot{u} = -\frac{\dot{u}}{\alpha} + s \cos \theta \]
\[ \dot{v} = -\frac{\dot{v}}{\alpha} + s \sin \theta \]
\[ \dot{u}' = -\frac{\dot{u}'}{\alpha} + \dot{\theta} \]
\[ \dot{v}' = -\frac{\dot{v}'}{\alpha} + \dot{\theta} \]
\[ \dot{z} = -\frac{\dot{z}}{\alpha} + s \cos (\theta - \zeta) \]
\[ \dot{\zeta} = \frac{1}{\alpha} s \sin (\theta - \zeta) \]
\[ \dot{z}' = -\frac{\dot{z}'}{\alpha} - s \cos \zeta' \]
\[ \dot{\zeta}' = \frac{1}{\alpha} s \sin \zeta' - \dot{\theta} \]  \hspace{1cm} (F.6)

Because these are still special cases of the more complex equations dealt with above, they still conform to the property of displaying the same systematic errors as each other.
Appendix G

Equivalence of Rotating Agent to Linearly Damped Pendulum

Below we show that a homing agent whose forward speed is held at zero can be well described by a three variable system having homology to a linearly damped pendulum. The direction of gravity is the same as the direction to the estimated nest location, with the magnitude of gravity increasing linearly with distance to the nest. This helps explain how the system can produce straight motion towards the nest from large distances, but allow complicated search patterns to develop at close range since “gravity” is not strong enough to hold the agent’s heading in a homeward direction.

We begin by defining simple linear neurons to represent the rotation control neurons of the agent. This is justifiable for the times when the sigmoidal neurons are within the linear part of their activation functions. Below $\theta$ is the agent’s heading, $u$ and $v$ are the states of the two linear neurons, and $(x, y)$ is the agent’s geocentric Cartesian home vector (constant, since forward speed is set to zero):

\[
\dot{\theta} = k(u - v)
\]
\[
\dot{u} = x \sin \theta - u + wv
\]
\[
\dot{v} = y \cos \theta - v + uw
\]

where we treat the neuron time constants as unity (since only the ratio to the turning rate $k$ is significant), and the contralateral weights connecting the rotation control neurons as of magnitude $w$. Defining a new variable, $z = u - v$, and noting $x = r \cos \theta$ and $y = r \sin \theta$:

\[
\dot{\theta} = k z
\]
\[
\dot{z} = r \sin(\theta - \gamma) - (w + 1)z
\]

Noting $\theta - \gamma = -\gamma' \pm \pi$ and $r' = r$:
\[ \dot{z} = r' \sin \gamma' - (w + 1)z \]

and noting \( \dot{\gamma} \) is constant:

\[ \dot{\gamma}' = -kz \]

implying:

\[ z = -\frac{1}{k} \gamma' \]

\[ \gamma' = -k\dot{z} \]

finally giving:

\[ \gamma' + (w + 1)\gamma' + kr' \sin \gamma' = 0 \]

which is homologous to the equation for a linearly damped pendulum where \( \gamma' \) is the angular displacement of the pendulum from vertical (downwards), \((w + 1)\) is the linear damping constant and \(kr'\) is the strength of gravity divided by the length of the pendulum (Strogatz, 1994). \( \gamma' \) therefore has an unstable equilibrium at \( \pm \pi \) and an otherwise globally stable equilibrium at zero.