Active Shape Discrimination with Physical Reservoir Computers

Chris Johnson[1], Andrew Philippides[1] and Philip Husbands[1]

[1]Centre for Computational Neuroscience and Robotics
University of Sussex
cj82@sussex.ac.uk

Abstract

We present the first example of 'minimally cognitive' sensorimotor behaviour arising from a body as physical reservoir. By revisiting an experiment introduced by Beer (1996) and replacing the continuous-time recurrent neural network (CTRNN) therein with networks of mass-spring-dampers we demonstrate that bodies may be exploited for more than control and pattern generation and take over some tasks which were previously thought to require a central nervous system.

Introduction

The importance of embodiment in both generating and understanding adaptive behaviour has been increasingly recognised over recent years (Pfeifer and Bongard, 2007). This has resulted in a renewed focus on the form and function of the body. The exploitation of inherent, often passive, dynamics has demonstrated that there is much to be gained, in terms of efficiency and simplification of control, when body-brain-environment interactions are balanced and harmonious (McGeer, 1990; Iida and Pfeifer, 2004; Shim and Husbands, 2012; Zhao et al., 2013). Pfeifer and Iida (2005) have coined the term morphological computation to refer to the way in which a judiciously selected body morphology can be shown to simplify the task of a controller and might be considered to be 'doing' the computational work it had rendered unnecessary. An interesting, and as yet underexplored, extension of this line of thought is to consider how much explicit and active information processing the body might be capable of, further blurring the line between the nervous system and the body. In fact it has already been shown (Valero-Cuevas et al., 2007) that the tendon network of a human finger performs joint torque mode selection in response to varying ratios of tendon tensions: a biological example of explicit morphological computation in action.

This paper describes research intended as a first step towards exploring the information processing potential of networks of simplified muscle-like units acting within an embodied agent engaged in adaptive behaviour. In this work we follow Hauser et al. (2011, 2012), who have reframed morphological computation in compliant bodies as a branch of reservoir computing (Maass et al., 2004; Lukoševičius and Jaeger, 2009). Hauser et al. (2011) presented networks of mass-spring-damper elements, and showed that with the addition of a simple linear readout these spring networks can perform complex computation requiring non-linear transformation and integration, such as the approximation of filters and inverse kinematics for robot control. These particular networks are of especial interest because they are physically realisable and because of their similarity to various biomechanical muscle models (Hill, 1938; Seyfarth et al., 2002; Baratta and Solomonow, 1990).

In Hauser et al. (2012) it was further shown that when the model was extended to include a feedback loop the networks could also be trained to perform pattern generation without the need for external stimulation. Nakajima et al. (2013) extended the spring network to a biologically-inspired 3D structure and it was shown that this body could also approximate filters and generate limit cycles. Finally, Zhao et al. (2013) replaced the spring network with the body of a spine-driven quadruped robot, referred to as 'Kitty', and used it to generate both locomotion and its own control signals. This robot stands out because the reservoir consists of force sensors embedded within the spine, the element of the body which is actuated, thereby negating any meaningful distinction between body and control.

In the above examples, morphological computation has been demonstrated to make difficult problems such as locomotion both easier and cheaper. However, filtering, pattern generation, and gait control have a character which is more automatic than cognitive. For example, although different gaits may be programmed into Kitty it is still essentially an automaton - its gait may be robust to some variation in the environment but it is incapable of responding to any stimuli which do not reach its force sensors.

We present the first example of 'minimally cognitive' sensorimotor behaviour arising from a body as physical reservoir. We revisited an experiment introduced by Beer (1996) where an agent controlled by a continuous-time recurrent neural network (CTRNN) was shown to be capable of discriminating between objects of different shapes through active perception. We replaced the CTRNN with a pair of networks based on those introduced by Hauser et al. (2011) and used an evolutionary algorithm to search for valid controllers.

This section has given the theoretical background and introduced the experiment reported upon here. The next section will describe the experiment and the spring network implementation. We will then describe the results obtained before closing with discussion of the results and some future directions for the work.

Methods

Methods are described in three subsections. First the experiment is described, in terms of the agent-environment interaction. Secondly the spring network, the agent's computational core, is described. Finally details of the evolutionary search for valid controllers are given.

The experiment

The simulated experiment is closely based on that described by Beer (1996, 2003). The required behaviour is to dynamically discriminate between a circular object and a diamond-shaped object. Discrimination is manifested as catch and avoidance behaviours for circles and diamonds, respectively. The arena is an area of 400 x 275. Circular objects are of diameter 30 and diamonds have side length 30.

Objects fall straight down from the top of the arena towards the agent with speed 3. In theory both behaviours are tested for at 24 equispaced points in the x-axis interval $[-50, 50]$. However, the use of a symmetrical controller means that only the left-hand 12 tests need be conducted as behaviour on the right-hand side is identical to that on the left. The agent has an antagonistic motor pair aligned to the horizontal axis. The network outputs set the two motor speeds, and the agent's velocity along its axis is the sum of the two opposing motor outputs. The transfer function for the motor pair is given by:

$$5(\sigma(s_r + \theta_r) - \sigma(s_l + \theta_l)) \tag{1}$$

$$\sigma(x) = 1/(1 + e^{-x}) \tag{2}$$

Due to the use of the logistic function σ, each motor saturates at 0 for its minimum and 1 for its maximum. This and the use of a multiplier of 5 for the result of the sum specifies a horizontal velocity in the range of $[-5, 5]$. θ is a constant value which biases the motor activation point.

The agent's sensors are 7 rays uniformly spaced across an angle of π/6 and centred about the vertical axis. The sensor transfer function is an inverse linear one between the distances of 220 and 0, with its output in the range [0, 10]. Objects are not detected beyond distances of 220. To reduce evaluation time the sensor model was used to construct lookup tables which were then used in the simulation. The sensor neuron activations lag behind the values of the linear function, as determined by the sensory layer function:

$$\tau_i \dot{s_i} = -s_i + I_i(x, y) \qquad i = 1, \ldots, 7, \tag{3}$$

where s is the sensor neuron activation, τ is the time constant for the sensor response, I is the sensor function, and (x, y) is the distance vector from sensor to object.

Network states, sensor activations and the position of the agent are all integrated using the forward Euler integration. As in Beer's original experiment an interval of 0.1 is used to integrate sensor activations and the agent's position. In their experiments Hauser et al. used an interval of 0.001 and made use of a solver function to integrate the spring network activity. However the computational cost of such an approach is problematic when evaluating large numbers of candidate controllers in evolution, so a compromise was made here. We found that an interval at least as small as 0.01 is required to achieve stability in the spring model with the parameters used here, so the spring network is integrated 10 times for each 0.1 interval.

Spring networks

Although the elements in these networks are in fact modelled mass-spring damper systems, for the sake of convenience they will henceforth be referred to simply as springs.

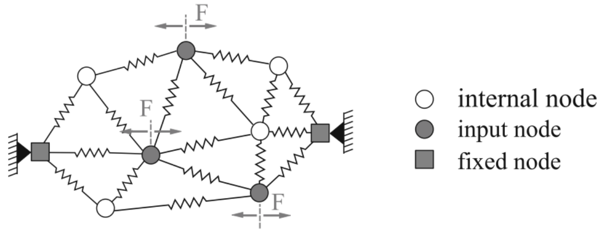

Figure 1: A spring network. The nodes at the opposite ends in the x-axis are fixed while the others are free to move. Some or all of those nodes receive an input as a force in the x-axis. Adapted from Hauser et al. (2011)

The spring networks used here are based upon those in Hauser et al. (2011), illustrated in Fig. (1). The springs are connected to each other in a 2-dimensional plane. Effects such as gravity and friction are neglected in order to simplify the model. The two outermost nodes in a selected axis are fixed while the rest move freely. A subset of the free nodes receive inputs in the form of applied forces. Input forces are applied in a single axis, although this is also for simplification and is by no means a requirement of the model. Reservoir elements were modelled as non-linear springs, defined by the state equations:

$$\dot{x_1} = x_2 \tag{4}$$

$$p(x_1) = k_3 x_1^3 + k_1 x_1 \quad (5)$$

$$q(x_2) = d_3 x_2^3 + d_1 x_2 \quad (6)$$

$$\dot{x_2} = -p(x_1) - q(x_2) + u, \quad (7)$$

where x_1 is the spring extension, k_1 and k_3 represent linear and non-linear stiffness coefficients, respectively, d_1 and d_3 represent the corresponding damping coefficients, and u represents an input force unused in this experiment. In this work we followed the network model of Hauser et al. (2011) in all respects except that the above nonlinear spring model was not used in all networks. In some networks a linear second order spring model was used, with the state equations:

$$\dot{x_1} = x_2 \quad (8)$$

$$\dot{x_2} = -\frac{k}{m}x_1 - \frac{d}{m}x_2 + \frac{1}{m}u, \quad (9)$$

where k is a stiffness coefficient, d is a damping coefficient, m is the mass on the end of the spring, and, as in Eq. (7), u is an unused input term. For convenience all nodes are given $m = 1kg$. This means that, from Newton's second law of motion, $F = ma$, forces and accelerations may be treated as equivalent in this network model and Eq. (9) is simplified to a form similar to Eq. (7).

At the beginning of each simulation step the spring extensions are obtained by calculating the distances between the nodes they connect. The rates of change of spring extensions are estimated by the difference between the current extensions and those at the previous step. From these states the instantaneous forces applied to the nodes by the springs can be found, by the use of either Eq. (7) or Eq. (9). The spring forces and input forces are then summed for each node, and the node positions are updated by integration of the resultant accelerations.

Inputs are applied to nodes as horizontal forces, as shown in Fig. (1). In this experiment each network had a total of nine nodes, with two fixed nodes and seven free nodes which each received an input from one of the sensor neurons (see Fig. (2)). An untreated input range of $[0, 10]$ from the sensor neurons was found to give poor results, and so the sensor neuron outputs were scaled and shifted to be in the range $[-0.5, 0.5]$.

The spring network output is a weighted sum of the extensions and extension rates of change of all springs in the network. There is a small departure from Hauser et al. (2011) here. We use the spring extension in the output sum where they used the overall length. The outputs of the two networks are fed into the motor function Eq. (2) in the same way as the CTRNN motor neuron outputs were in Beer (1996).

The CTRNN controller in Beer (1996) was bilaterally symmetric. In this case symmetry of control is achieved by having two identical networks of springs, one of which receives its inputs from the sensory neurons in the reverse order to the other. The CTRNN controller consisted of a layer of five fully interconnected recurrent interneurons, and two feedforward motor neurons. The spring network pair replaces these seven neurons.

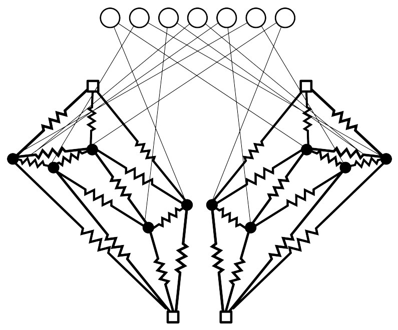

Figure 2: The symmetrical network pair. Two identical networks are connected to the sensor array in opposite orders. The outputs of the two networks are linear sums of the spring states, converted into agent velocity by the use of Eqn. (2). Some nodes and springs are omitted from this diagram for clarity.

Searching for solutions

Some network parameters are generated randomly and others are set by a search with a Macroevolutionary Algorithm (MA) (Marin and Sole, 1999). The MA was selected over a Genetic Algorithm (GA) because it was found to be less prone to premature convergence to local optima in the search space for this task. The algorithm was implemented as described in (Marin and Sole, 1999), except for the setting of the two dynamic constants, ρ and τ. The genetic radius for reproduction, ρ, was set by the function $\rho = 0.3(1 - f_{max})$, subject to a minimum value of $\rho = 0.1$. The temperature parameter, τ was set by the function $\tau = (1 - f_{max})$, subject to a minimum value of $\tau = 0.2$. In addition, a constraint was set such that at least one of each generational offspring was randomly generated, in order to promote diversity in later stages. For the same reason, a mutation operator was added such that on average one gene per genotype would be mutated. Mutated genes are moved by an amount in the range of $\pm 10\%$ of the total genetic interval with a probability of 0.9, and replaced with a random value with a probability of 0.1.

A single network topology generated at random at the beginning of each run of the MA is employed by all members of the population. The node coordinates are generated randomly in an area 10 x 10, and then connected with springs by the use of a Delaunay triangulation (Lee and Schachter, 1980). The use of this triangulation method tends to maximise the triangle angles, but also leads to a variable number of springs in the network. Parameters which may be determined by the search are: spring coefficients for stiffness and damping, weights on the sensory inputs to the networks, weights on the spring states for the linear readout, feedback gains, and the bias term for the motor function in Eqn. (2).

For the purpose of evaluation the horizontal distance, d_i, between the agent and the object is clipped to a maximum of 45 and then normalised between 0 and 1. For a catch trial the controller scores $1 - d_i$ and for an avoid trial the score is equal to d_i. The final score for a controller is the mean of its individual trial scores. The horizontal distance is clipped to prevent success in one behaviour from dominating a controller's score at the expense of the other.

The MATLAB IDE (The Mathworks, Inc., Natick, MA) was used for all aspects of agent simulation, evolution, and later analysis.

Results

Results are presented in the following order: first, details are given of the performance of the evolutionary search for valid controllers. Secondly, we show some of the inner workings of the network and briefly examine its capacity for memory. We end this section with analysis of two networks by viewing the impact on performance of various lesions.

Searching for solutions

A set of controller features may be enabled or disabled at the beginning of each evolutionary run. These features are: whether to use the linear or non-linear spring model, whether to use spring velocity in the linear readout, whether to evolve real-valued weights on the inputs, whether to use a single random set of spring parameters across the population or to evolve those parameters, whether to employ node position feedback and whether to evolve or to use a constant value for the motor bias in Eqn. (2). The ranges of all evolved parameters are given in Table. (1).

Parameter	Upper limit	Lower limit
Position	10000	-10000
Velocity	10000	-10000
Input weights	-2	2
Linear stiffness coefficients	1	100
Linear damping coefficients	1	100
Non-linear stiffness coefficients	100	200
Non-linear damping coefficients	100	200
Motor function bias	-5	5
Feedback gains	-1	1

Table 1: Limits placed on evolved values.

When node position feedback is employed it is applied as an xy force vector based on a node's displacement from its resting position. Where the motor bias is not evolved a constant value of 2.5640, taken from a successful CTRNN controller which was found when developing the simulation, was used. Where input weights are not evolved, one set of weights is randomly drawn from the set $\{-1, 1\}$, and applied to the entire population. These unity weights are of both signs in order to avoid the entire network being pushed in a single direction. When spring parameters were not evolved they were randomised as reported in Hauser et al. (2011). Non-linear spring coefficients are drawn from a uniform distribution in the interval $[100, 200]$. Linear spring coefficients are drawn from a log-uniform distribution in the interval $[1, 100]$. The use of the log-uniform distribution biases samples towards the lower end of the interval.

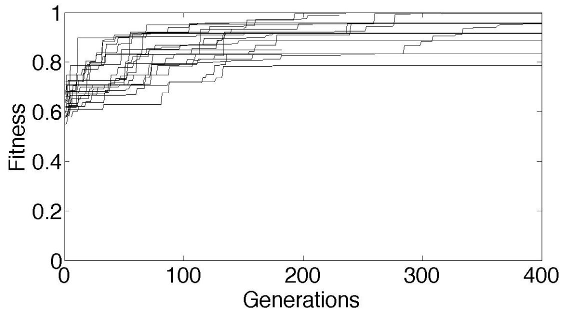

Figure 3: Performance of the MA for a configuration matching controller A, over evolutionary runs of up to 700 generations with a population size of 400. Runs were cancelled when they were either successful or effectively halted. Only the first 400 generations are shown here.

It was initially unclear which configuration of features was most appropriate, so a set of 20 evolutionary runs, each with a single randomly generated configuration applied across the population, was executed. A population of size 400 was evolved over a short run of 100 generations. It should be pointed out that the MA favours larger populations, but also that the number of the population replaced, and therefore requiring evaluation, upon each generation is variable and typically much less than the population size.

Controller	A	B	C	D	E
Fitness(%)	99.6	98.9	99.5	98.4	97.8
Number of springs	20	18	18	18	18
Velocity	1	1	1	0	1
Weighted inputs	1	0	1	1	1
Evolve springs	1	0	0	0	1
Nonlinear springs	1	0	0	1	1
Bias motors	1	0	0	0	0
Feedback	0	1	0	1	1

Table 2: Winning combinations. The features of velocity in the readout sum, weighted inputs, evolved springs, nonlinear springs, evolved motor function bias and node position feedback are all optional. 20 evolutionary runs of 100 generations with a population size of 400 were run with random selection of optional features. 5 runs generated successful controllers; each with a unique configuration.

A success threshold of ~98% of the perfect score was

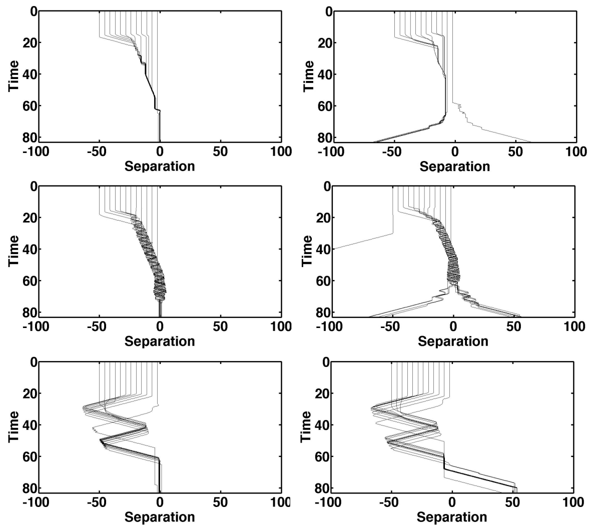

Figure 4: Agent trajectories throughout trials. Trajectories from catch trials are shown on the left and those from diamond trials on the right. Trajectories from three controllers are shown. From top to bottom, controllers A, C and A1.

determined to be sufficient to ensure the correct behaviour, and 5 out of 20 runs resulted in viable controllers. Table. (2) shows the configurations of these 5 controllers. Although this is a small set of results, the variety is striking - the configurations of controllers A and E are similar, but otherwise there is no evidence of a particular configuration which is required for success. However it can be seen that the use of velocity in the linear readout is favoured; being used in 4 out of 5 controllers. Of the remaining features only whether or not to evolve the motor bias stands out; selected in only one result.

Following these results a further 20 evolutionary runs were executed with the arbitrarily selected configuration of controller A. In this case runs continued until the search could be seen to either have succeeded or effectively halted at a local optimum. In this case 4 runs succeeded although others came close. Fig. (3) shows the progress of these runs across the first 400 generations. As yet it is unclear as to whether the difficulty of the problem or the character of the MA is more responsible for the number of failures. However, since evolutionary search is merely being used as a method to find a viable solution, and it readily finds several in our batch runs, its efficiency is not a major concern at present. One of this second set of results, referred to as A1, appears in figures and analyses throughout this section.

Successful networks have proven to be diverse in their topologies as well as their configurations. As shown in Fig. (5) success does not seem to require any particular form of network.

Finally, from a high-level point of view, it can also be seen that various different strategies are possible. The trajectories of controller A, C and A1 for all 24 trials are shown in Fig. (4). Controller A inches towards objects until it can distinguish between them, controller C finds objects quickly and then oscillates around their position until making a decision, and controller A1 scans back and forth.

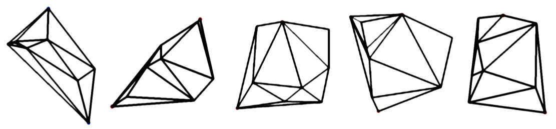

Figure 5: Network topologies of successful controllers. Controllers A through to E are shown from left to right.

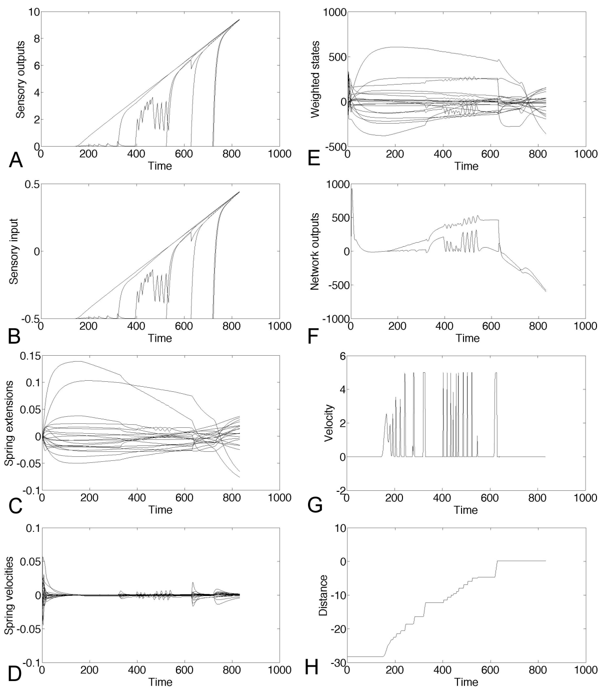

Figure 6: Plots of inputs, states and outputs of controller A from a catch trial. All plots are over time. Plot A shows the outputs of the sensory neurons. Plot B shows the shifted and scaled sensory input to the networks. Plot C shows the spring extensions of one network. Plot D shows the spring velocities of the same network. Plot E shows the weighted positions added to the weighted velocities. Plot F shows the outputs of both networks. Plot G shows the velocity of the agent. Plot H shows the distance between the agent and object in the x-axis.

Network analyses

Fig. (6) illustrates some of the workings of controller A in making a successful catch. As mentioned previously, the sensory inputs are scaled and shifted to the range $[-0.5, 0.5]$.

This has the effect of biasing the network to activity, with the springs already in motion for the first 10s even though the sensory neurons have zero output. The spring extensions and velocities are combined in the readout in this controller. Due to the very large weights used in the readout the network outputs are similarly high. Given that the motor function includes the saturating logistic function, this leads to a motor behaviour of rapidly switching the motors on and off, resulting in the agent creeping incrementally towards the position of the falling circle.

An interesting observation is that as long as the sensory neurons all have zero output, so do the networks. Given that the spring positions are non-zero in this initial period, it is clear that the linear readout is balanced to not respond to this quiescent activity. This is perhaps appropriate to reactive behaviour, but the possibility remains for evolution to lead to a more proactive search strategy by generating an imbalanced readout.

We next turned to measuring the memory of the network following a method based on that described by Maass et al. (2004). Maass et al devised an input stream with zero mutual information (and therefore also zero correlation) between different segments. Then, to obtain a measure of network memory, readout neurons were trained to reproduce segments of the input stream from earlier periods, and segments of the output signal were correlated against the input segments they should have reproduced. A similar approach is taken here, although we chose not to measure general memory capacity but rather to see if these networks can retain information of the inputs they are evolved to deal with. For this reason the readout was trained to recover the simplest combination of the input signals over the course of a single trial, their sum into a single time series. In this case many segments of the input stream have high correlation with one another as, due to the tendency of agents to position themselves under an object until it can be recognised, the general trend is for sensory input to increase as a ramp. Therefore, as a baseline, the same series of correlations was performed for input segments against one another. Where the readout shows no more correlation with earlier input stream segments than its corresponding input segment does, then there can be considered to be no memory. On the other hand, a stronger correlation between the output of the readout and the delayed input it is trained to reproduce than between input and delayed input is indicative of memory in the network. The same test is performed for readouts which receive only spring extensions as inputs, readouts which receive only spring velocities, and readouts which receive both.

The results of some of these tests are shown in Fig. (7). It can be seen that, in general, spring extensions encode more relevant information than spring velocities, but that the combination of the two encodes more than either alone. Controller A shows no convincing sign of memory. For the avoid

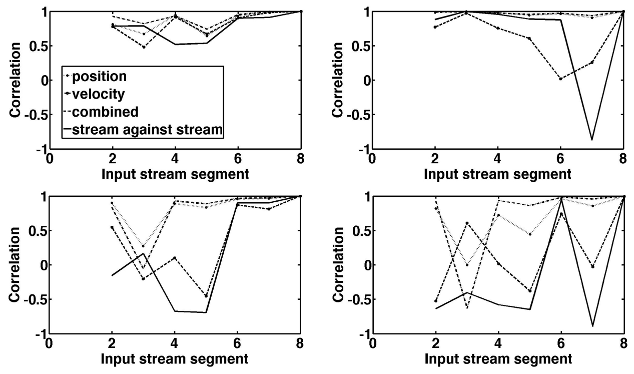

Figure 7: Measuring memory. The linear readout is trained to recover inputs from 10s ago. Then the input stream and the controller output are split into 10s segments and the correlation of each output segment with the prior input segment is calculated. Plots on the left are from a single catch trial and those on the right are from an avoid trial. The results for two controllers are shown. The top row is for controller A, and the bottom row is for controller A1.

trial there is a period where the correlation of input segment against input segment is of the opposite sign to that of readout segment against input segment, but the magnitudes are roughly equal. This seems consistent with the behaviour of this controller. As shown in Fig. (4) for both catch and avoid behaviours, initially this controller gradually creeps towards the object as it falls, suggestive of a purely reactive network. The result for controller A1, however, does indicate a degree of memory effect, with the network being able to recover more information about earlier input segments than the input stream itself. Once again, this is consistent with the general strategy - unlike other results this controller drives away from the object and then returns to it, a behaviour which seems of a more proactive character and implies memory of at least which side of the agent the object is on. It should be pointed out that this controller does not make use of feedback. Any present memory is only transient, fading memory.

Analysis of a network with such complex dynamics in a sensorimotor loop is far from trivial, but a certain amount can be discovered by recording changes in behaviour as parts of the network are disabled. Four experiments, illustrated in Fig. (8) for two controllers, were conducted. In the first three experiments changes were made to the springs, one at a time, and the performance scores for the modified network for all 24 trials were recorded. In the fourth, one input at a time was disabled and the performance scores were recorded. In order, the modifications for springs were: to disconnect them from the linear readout, to remove them from the network completely, and to disable their non-linearity.

Various observations can be made from these plots. It

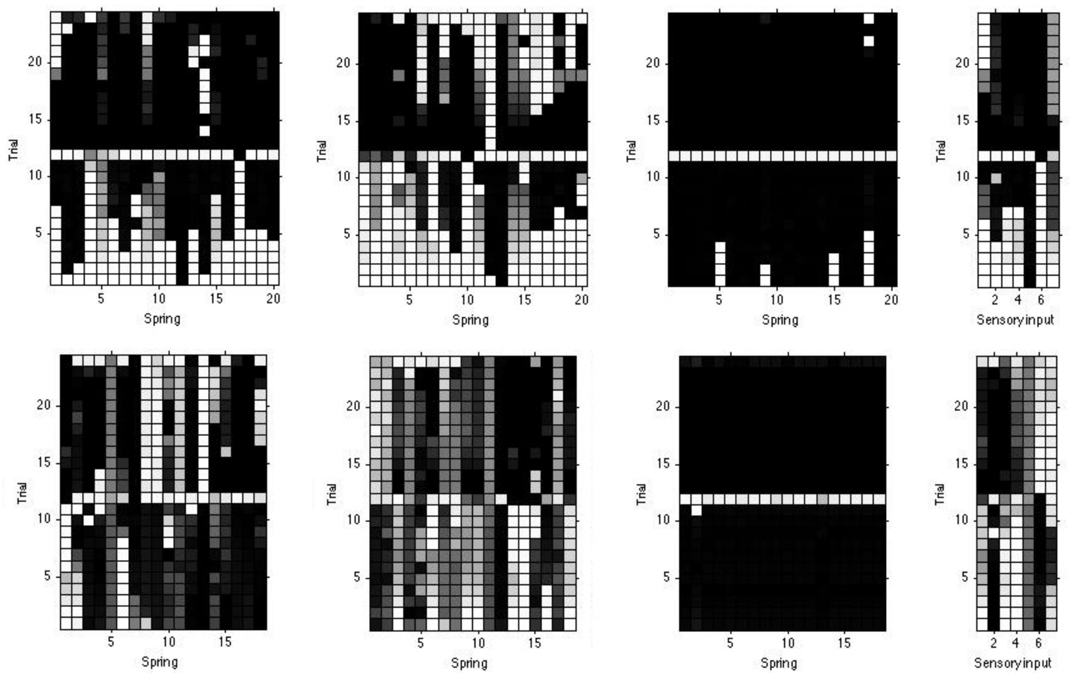

Figure 8: Lesion experiments. The top row of plots shows results for controller A and the bottom for controller A1. The rows in plots show the performance on a trial by trial basis. The brightness of a grid element shows the effect of the lesion. Black regions show unaffected performance and lighter regions show impaired performance. Trials 1 to 12 are catch trials and trials 13 to 24 are avoid trials. From left to right: one spring at a time is disconnected from the readout; one spring at a time is removed from the network, one spring at a time is linearised; one input at a time is disconnected.

can be seen that adjustments to the network for controller A tend to cause failure in catching circles far more often than in avoiding diamonds, as though this controller is predisposed to avoidance. To support this conclusion, this agent also shows low dependence on all but the outermost sensors for avoidance. Complete removal of springs from the network causes a lot of failure in both agents. This is no surprise, given the tightly coupled nature of the network dynamics. Neither controller shows a strong dependence on spring non-linearity,

The plots in Fig. (8) suggest that for both of these agents the most difficult trial is the last where catching behaviour is required. This is surprising as at the beginning of this trial the object is only slightly offset from the agent's position. The reason for this has not yet been properly uncovered, but it seems probable that it is connected to the large weights in the linear sum as relatively small differences in sensory input are amplified into high velocity, which could lead to a sudden loss of the object's position.

Discussion

Discussion of results

The CTRNN described by Beer (1996) was bilaterally symmetric and fully interconnected in the interneuron layer. Symmetry is achieved here by the use of a pair of identical networks, coupled only indirectly by their roles in the sensorimotor loop. The overall behaviour of the agent is therefore the result of complementary activity in the two sides of the simulated body.

Successful agents obtained by evolution pass the bar for autonomous task-based behaviour, capable of both responding directly to stimuli and making use of short-term memory to guide their motion. The behaviour of these agents goes beyond the maintenance of locomotive gaits to the selection of an appropriate action based on active integration of sensory information. This more complex behaviour is also achieved with a relatively small number of springs. Hauser et al. (2011) used a network of 78 springs; these controllers contain approximately 40.

The morphological computation evident in the tendon network of the human hand is a striking example of parsimony in evolution. This work lends support to the hypothesis that bodies can be and will be doing some of the computational work which used to be considered the sole dominion of the central nervous system.

The use of an evolutionary algorithm to obtain valid controllers has led to strong indications that the domain of these networks is rich in its variety of computational resources.

This potential for diversity may have been less apparent if a learning algorithm was used.

Magg and Philippides (2006) have already shown that this problem may be solved with a non-dynamical ANN, and, as in controller A, it appears that most results seen so far employ little or no memory. However, controller A1 is an interesting result which does seem to rely on memory and suggests that behavioural tasks of a more challenging character than this one may therefore also be addressed with similar spring networks.

Further work

The very large weights used in the linear readout sum tend to lead to the agent switching between extremes of velocity. In a real-world system this would be inefficient and probably lead to shortened motor life. At present we are examining how far the scale of those weights may be reduced while still achieving valid controllers. First results are encouraging, with signs of smoother behaviour.

The results obtained so far indicate that due to the tight coupling throughout the networks they will not fail gracefully when damaged, as indicated in Fig. (8). However, it may be that a damaged network may be easily retrained to recover its function. In future work we will examine the efficacy of retraining by submitting impaired controllers back to evolution.

Although enough good and varied results were found to convince that these paired networks may be used to effect reactive behaviour, efforts are underway to both improve and tune the evolutionary algorithm in use and to make the networks more evolvable. As far as the second point goes, one feature of the problem which has not yet been placed under evolutionary control is the network topology. We believe that topological design by evolution will lead to an improved success rate.

Later projects will explore the capability of spring networks to generate behaviour in compliant robots, and to deal with real-world, noisy, situations.

References

Baratta, R. and Solomonow, M. (1990). The dynamic response model of nine different skeletal muscles. *Biomedical Engineering, IEEE Transactions on*, 37(3):243–251.

Beer, R. D. (1996). Toward the evolution of dynamical neural networks for minimally cognitive behavior. *From animals to animats*, 4:421–429.

Beer, R. D. (2003). The dynamics of active categorical perception in an evolved model agent. *Adaptive Behavior*, 11(4):209–243.

Hauser, H., Ijspeert, A. J., Füchslin, R. M., Pfeifer, R., and Maass, W. (2011). Towards a theoretical foundation for morphological computation with compliant bodies. *Biological cybernetics*, 105(5-6):355–370.

Hauser, H., Ijspeert, A. J., Füchslin, R. M., Pfeifer, R., and Maass, W. (2012). The role of feedback in morphological computation with compliant bodies. *Biological cybernetics*, 106(10):595–613.

Hill, A. (1938). The heat of shortening and the dynamic constants of muscle. *Proceedings of the Royal Society of London. Series B, Biological Sciences*, 126(843):136–195.

Iida, F. and Pfeifer, R. (2004). Cheap rapid locomotion of a quadruped robot: Self-stabilization of bounding gait. In *Intelligent Autonomous Systems*, volume 8, pages 642–649.

Lee, D.-T. and Schachter, B. J. (1980). Two algorithms for constructing a delaunay triangulation. *International Journal of Computer & Information Sciences*, 9(3):219–242.

Lukoševičius, M. and Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149.

Maass, W., Natschläger, T., and Markram, H. (2004). Computational models for generic cortical microcircuits. *Computational neuroscience: A comprehensive approach*, pages 575–605.

Magg, S. and Philippides, A. (2006). Gasnets and ctrnns–a comparison in terms of evolvability. In *From Animals to Animats 9*, pages 461–472. Springer.

Marin, J. and Sole, R. V. (1999). Macroevolutionary algorithms: a new optimization method on fitness landscapes. *Evolutionary Computation, IEEE Transactions on*, 3(4):272–286.

McGeer, T. (1990). Passive dynamic walking. *the international journal of robotics research*, 9(2):62–82.

Nakajima, K., Hauser, H., Kang, R., Guglielmino, E., Caldwell, D. G., and Pfeifer, R. (2013). A soft body as a reservoir: case studies in a dynamic model of octopus-inspired soft robotic arm. *Frontiers in computational neuroscience*, 7.

Pfeifer, R. and Bongard, J. (2007). *How the body shapes the way we think: a new view of intelligence*. MIT press.

Pfeifer, R. and Iida, F. (2005). Morphological computation: Connecting body, brain and environment. *Japanese Scientific Monthly*, 58(2):48–54.

Seyfarth, A., Geyer, H., Günther, M., and Blickhan, R. (2002). A movement criterion for running. *Journal of biomechanics*, 35(5):649–655.

Shim, Y. and Husbands, P. (2012). Chaotic exploration and learning of locomotion behaviors. *Neural computation*, 24(8):2185–2222.

Valero-Cuevas, F. J., Yi, J.-W., Brown, D., McNamara, R. V., Paul, C., and Lipson, H. (2007). The tendon network of the fingers performs anatomical computation at a macroscopic scale. *Biomedical Engineering, IEEE Transactions on*, 54(6):1161–1166.

Zhao, Q., Nakajima, K., Sumioka, H., Hauser, H., and Pfeifer, R. (2013). Spine dynamics as a computational resource in spine-driven quadruped locomotion. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1445–1451. IEEE.