Active Shape Discrimination with Compliant Bodies as Reservoir Computers

Chris Johnson*,**
Andrew Philippides**
Philip Husbands**
University of Sussex

Keywords
Morphological computation, reservoir computing, adaptive behavior, active perception, minimal cognition

Abstract Compliant bodies with complex dynamics can be used both to simplify control problems and to lead to adaptive reflexive behavior when engaged with the environment in the sensorimotor loop. By revisiting an experiment introduced by Beer and replacing the continuous-time recurrent neural network therein with reservoir computing networks abstracted from compliant bodies, we demonstrate that adaptive behavior can be produced by an agent in which the body is the main computational locus. We show that bodies with complex dynamics are capable of integrating, storing, and processing information in meaningful and useful ways, and furthermore that with the addition of the simplest of nervous systems such bodies can generate behavior that could equally be described as reflexive or minimally cognitive.

1 Introduction

The importance of embodiment in both generating and understanding adaptive behavior has been increasingly recognized over recent years [6, 7, 28, 29]. This has resulted in a renewed focus on the form and function of the body. Repeated successes in the exploitation of inherent, often passive dynamics in automata and robots have demonstrated that much can be gained, in efficiency and simplification of control, when body–brain–environment interactions are balanced and harmonious [24, 15, 32, 36, 31]. Pfeifer and Iida [30] introduced 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 therefore be considered to be performing a function analogous to the computational work it renders redundant. 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. This article 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 behavior. In this work we follow Hauser et al. [11, 12], who have reframed morphological computation in compliant bodies as a branch of reservoir computing (RC) [22, 20], an approach they demonstrated with simulations of recurrent networks of mass–spring–damper (MSD) elements.

* Contact author.
** Centre for Computational Neuroscience and Robotics, University of Sussex, Brighton, BN1 9RH, U.K. E-mail: c.a.johnson@sussex.ac.uk (C.J.); andrewop@sussex.ac.uk (A.P.); p.husbands@sussex.ac.uk (P.H.)

The passive dynamic walker [24] is an early exemplar of what later became known as morphological computation. McGeer's automaton was capable of walking down a gentle incline under the sole force and control of gravity by means of a pendulum-based design. While the walker can only perform one action, and that only in a narrow niche, it still stands out in that it exhibits action entirely without control. Since the notion of morphological computation was introduced, the majority of robotic studies in this area have been in a similar vein and have focused on minimization of control and increase in robustness via the exploitation of carefully designed bodily dynamics. Often this is achieved by designing the robot morphology to be both compliant and self-stable [15, 30, 31].

An issue still at large is whether what is referred to as morphological computation should properly be considered computational. The basis of morphological computation is not formal logic, nor, except in special and rare cases [27, 33], does it operate in a binary domain. It is in no way in keeping with Turing's definition of what is computable and has no affinity with classical automata theory [10]. However, running parallel to what has, perhaps irreversibly, become the mainstream of computing is a tradition of analogue computing [21, 18]. For decades, analogue computation was the only feasible option for the simulation and control of complex systems in applications such as aeronautics and space flight [18]. In general, analogue electronic computers are dynamical systems and therefore suitable for the simulation and control of other dynamical systems. We believe that morphological computation, as first presented by Pfeifer and Iida [30], is of the same kind and so legitimized as computational as long as it is clear which tradition we refer to. In fact the name "analogue electronics" is derived from analogy, due to the isomorphism between a mass–spring–damper system and an *RLC* electronic circuit (a circuit including a resistor, capacitor, and inductor) [18, 1]. Therefore there is a direct connection between an MSD network and the very origin of analogue computing. In the remainder of this article, where we use the word "computation," we will be referring to analogue computation.

Hauser et al. [11] presented networks of mass–spring–damper elements, and showed that with the addition of a simple linear readout, theoretically consistent with reservoir computing, these spring networks can perform complex computation requiring nonlinear transformation and integration, such as the approximation of filters and inverse kinematics for robot control. These networks are of special interest because they are physically realizable and because of their similarity to biomechanical muscle models [13, 9, 2].

In Hauser et al. [12] it was further shown that when the model was extended to include a feedback loop, the networks could be trained to perform pattern generation without the need for external stimulation. Nakajima et al. [26] extended the spring network to a biologically inspired three-dimensional structure, and it was shown that this body could also approximate filters and generate limit cycles. Finally, Zhao et al. [36] 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 that 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 that is more automatic than intelligent. 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 that do not reach its proprioceptive force sensors.

We will show that morphological computation can go further. If continuous-time neural networks (CTRNNs) [35], GASNets [14], and other dynamical neural models can generate adaptive behavior, then why not these dynamical networks also? And if these networks, constructed of muscle-like units, can do so, then we are led to interesting speculations about the extent of muscle properties in determining behavior, rather than merely being in the service of a central controller.

In order to test whether MSD networks can generate adaptive behavior, we selected an experiment introduced by Beer [3], in which an agent controlled by a small CTRNN was shown to be capable of discriminating between objects of different shapes through active perception. We replaced the bilaterally symmetric CTRNN used by Beer with a symmetrical pair of MSD networks. Previously the minimal cognitive aspect of the task had been made much of [4, 8, 25]. We consider this a point

well enough made, and approach the experiment from a specifically morphological computation point of view. The MSD network is essentially an abstracted passive compliant body, and the readout can be interpreted as analogous to either a primitive or a peripheral nervous system. In this context we consider the behavior exhibited by our agents as being of the reflexive variety, with the choice made by the agent being to either initiate escape behavior or not, in response to different patterns of stimuli.

In the rest of the article we show that we can evolve MSD networks that are capable of generating adaptive behavior, and through a number of analyses we show that, in principle, bodies are capable of integrating, storing, and processing information in meaningful and useful ways. In what follows it will be made clear that the domain of MSD networks affords a broad variety of behaviors for this task and that the evolutionary process exploits an informational structure in the agent's visual field. However, we will also demonstrate that a distinction must be made between the quantity and quality of information. We will close with an examination, using perturbation and lesion tests, of how the networks actually perform their function, and with case studies of two controllers that generate behavior that is representative of the two far ends of the observed behavioral spectrum.

2 Methods

2.1 The Experiment

The simulated experiment is closely based on that described by Beer [3, 4]. The required behavior is to dynamically discriminate between a circular object and a diamond-shaped object. Discrimination is manifested as catch and avoidance behaviors for circles and diamonds, respectively (see Figure 1 for a depiction of the agent and its controller).

The arena is a rectangular area 200 wide by 275 high. 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 behaviors are tested 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 behavior on the right-hand side of the arena is identical to that on the left. The agent has an antagonistic motor pair aligned with the horizontal axis. The network outputs set the two motor speeds, and the agent's horizontal velocity is the sum of the two opposing motor outputs. The transfer function for the motor pair is given by

$$5[\sigma(N_r + \theta) - \sigma(N_l + \theta)] \tag{1}$$

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

where N_l and N_r are the outputs from the left and right MSD network readouts, respectively, and θ is a constant that biases the motor activation points. Due to the use of the logistic function σ, each motor saturates at 0 for its minimum and 1 for its maximum. This, together with the use of a multiplier of 5 on the result of the sum, specifies a horizontal velocity in the range of $[-5, 5]$.

The agent's sensors are seven rays uniformly spaced across an angle of $\pi/6$ and centered about the vertical axis. A sensor is activated if the sensor ray intersects an object. The sensor transfer function, I, 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 vector from sensor to object.

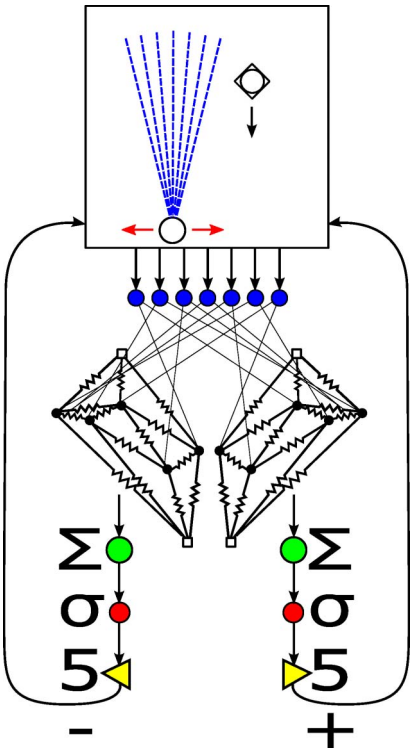

Figure 1. The agent–environment coupled dynamical system. The agent in the environment, including falling objects, is shown in the panel, and the connected controller is shown below. The agent moves left and right, to catch objects that are circle-shaped and avoid those that are diamond-shaped. Falling objects are detected by seven sensor rays. Seven sensor neurons each receive an input from a single sensor, and output to a single node in the MSD network. The weighted sum, Σ, of the network spring extensions, and in some cases also the spring extension velocities, represents a network's output. That output is passed through a motor neuron with a sigmoidal transfer function, σ, followed by a gain of 5. An identical pair of networks receive their inputs from the sensor neurons in reverse order to one another, and the output of each drives one of an antagonistic pair of motors. Note that the circle-shaped object, shown superimposed on the diamond-shaped object, is narrower than its counterpart from the agent's point of view.

Network states, sensor activations, and the position of the agent are all integrated using 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 prohibitive when evaluating large numbers of candidate controllers in evolution, so a compromise was made here. We found that an interval of at most 0.01 is required to achieve stability in the network model with the parameters used here, so the spring network is integrated 10 times for each 0.1 interval.

2.2 Mass–Spring–Damper Networks

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

The spring networks used here are based upon those in Hauser et al. [11] (Figure 2). The springs are connected to each other in a two-dimensional plane. Effects such as gravity and friction are neglected in order to simplify the model. The two outermost nodes on 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 along a single axis, although this is also for simplification and is by no

means a requirement of the model. Reservoir elements were modeled as nonlinear springs, defined by the state equations:

$$x_2 = \dot{x}_1 \tag{4}$$

$$p(x_1) = k_3 x_1^3 + k_1 x_1 \tag{5}$$

$$q(x_2) = d_3 x_2^3 + d_1 x_2 \tag{6}$$

$$\dot{x}_2 = -p(x_1) - q(x_2) + u \tag{7}$$

where x_1 is the spring extension, k_1 and k_3 are linear and nonlinear stiffness coefficients, respectively, d_1 and d_3 are the corresponding damping coefficients, and u represents an input unused in this experiment. In this work we followed the network model of Hauser et al. [11] 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

$$x_2 = \dot{x}_1 \tag{8}$$

$$\dot{x}_2 = -\frac{k}{m} x_1 - \frac{d}{m} x_2 + \frac{1}{m} u \tag{9}$$

where k is a stiffness coefficient, d is a damping coefficient, m is the mass on the end of the spring, and, as in Equation 7, u is an unused input term. For convenience all nodes are given $m = 1$ kg. 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 Equation 9 is simplified to a form similar to Equation 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 Equation 7 or Equation 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 vertical forces, as shown in Figure 2. 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

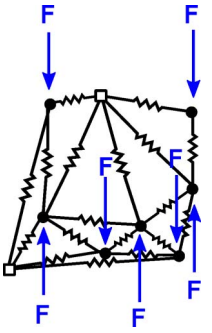

Figure 2. An MSD network. The nodes of the network are point masses, connected by parallel springs and dampers. The two most distant nodes, marked with squares, are fixed, while other nodes are free to move. Inputs to the network take the form of forces applied to all or a subset of the free nodes, parallel to the vertical axis. (Dampers and some springs and nodes are omitted for clarity.)

sensor neurons (Figure 1). 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 of all springs in the network, and in some cases also the extension velocities. There are two departures from Hauser et al. [11] here. We use the spring extension in the output sum, where they used the overall length, and we added extension velocities as a computationally cheap way to potentially increase the information encoded in the network output. The outputs of the two networks are fed into the motor function (2) in the same way as the CTRNN motor neuron outputs were in Beer [3]. The CTRNN controller in Beer [3] 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.

2.3 The Evolutionary Search Algorithm

Some network parameters are generated randomly, and others are set through a search with a macroevolutionary algorithm (MA) [23]. 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. In short, whereas a GA models microevolution, with individual pitted against individual, the MA models macroevolution, at the level of species. Each member of the population of the MA is a species, and selection proceeds not just based on fitness, but also on the similarity between the species. Essentially, the more similar a pair of species are, the more they are in competition. On every generation a matrix is built of the scores of all members of the population against each other, using the following function:

$$W_{i,j} = \frac{f(p_i) - f(p_j)}{|p_i - p_j|} \tag{10}$$

where $W_{i,j}$ is the score for individual i against individual j, $p_i = (p_i^1, \ldots, p_i^d)$ are the genes of the ith individual, and $f(p_i)$ is the fitness of the phenotype that is mapped from p_i. The overall score for a species is the sum of its scores against all other species, and all species with a negative total score become extinct. This means that the number of evaluations per generation is dynamic; often as small a proportion as around $\frac{1}{4}$ of the population will be selected for replacement.

With probability τ, an extinct species p_i is replaced with a randomly generated one. Otherwise it is replaced with a species recombined from the genes of the extinct species and a randomly selected survivor, p_b. The function for replacement by recombination is

$$p_i(t+1) = p_b(t) + \rho\lambda[p_b(t) - p_i(t)] \tag{11}$$

where $\lambda \in [-1, 1]$ is selected randomly, and ρ sets the radius around the extinct species in parameter space for placement of the new species.

The algorithm was implemented as described in [23], except for the setting of the two dynamic constants, ρ and τ. The genetic radius for reproduction, ρ, was set by the function $\rho = 0.3(1 - f_{\max})$, where $f_{\max}$ is the current maximum fitness, 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 a random amount drawn from a uniform distribution in the range of ±10% of the total genetic interval with a probability of 0.9, and replaced with a random value from a uniform distribution across the entire range of the gene 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 10×10, and then connected with springs by the use of a Delaunay triangulation [19]. The use of this triangulation method tends to maximize the triangle angles, but also leads to a variable number of springs in the network. Parameters that 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 Equation 2. All parameters are evolved as double-precision real numbers in the interval [0, 1] and scaled to appropriate values in the genotype-to-phenotype mapping. Genotype length is highly variable, as will be explained in Section 3.

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 normalized between 0 and 1. Hence for a catch trial the controller scores $1 - d_i$, and for an avoid trial 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 behavior 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.

3 Results

In this section we will look more closely at the means of setting the parameters of the networks and their readouts. In the reservoir computing paradigm, network parameters are typically generated randomly and only readout weights are adapted. However, because the networks used here are much smaller than is usual, it was not immediately obvious whether the same template should be followed here or whether all available parameters should be placed under evolutionary control. We will begin by describing the options we built into the process to be able to find an answer to this question, and then follow with a record of refinements, which ultimately led to a much improved success rate.

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 nonlinear spring model, whether to use spring extension velocity in the linear readout, whether to evolve real-valued weights on the inputs or to use random integers, 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 Equation 2 (Table 1, column 1). The ranges of all evolved parameters are given in Table 2. Due to the use of different configurations of which features to evolve, and because the use of the Delaunay triangulation leads to a variable number of MSDs in the network, the genotype length is highly variable. Approximate lengths for some configurations that led to a number of successful controllers are given in Table 3.

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 that was found when developing the simulation, is 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 unit weights are of both signs in order to avoid the entire network being pushed in a single direction. When spring parameters are not evolved, they are randomized as reported in Hauser et al. [11]. Briefly, nonlinear spring coefficients are drawn from a uniform distribution in the interval [100, 200], and 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.

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. While the MA favors larger populations, note that the proportion of the population replaced, and therefore requiring evaluation, is variable and typically much less than the population size. A success threshold

Table 1. The first winning combinations discovered.

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
Evolved 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

Notes. 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. Five runs generated successful controllers, each with a unique configuration.

of ≈98% of the perfect score was determined to be sufficient to ensure the correct behavior for all trials, and 5 out of 20 runs resulted in viable controllers. Table 1 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 that is required for success. However, it can be seen that the use of velocity in the linear readout is favored, being used in 4 out of 5 controllers. Of the remaining features, only whether or not to evolve the motor bias stands out, being selected in only one result.

Table 2. Limits placed on evolved parameters for configurations A through E.

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

Table 3. Statistics of evolutionary runs for configuration A and derived configurations.

Configuration	A	J	S	W
Number of genes (approx.)	120	120	26	160
Mean fitness	0.93	0.95	0.95	0.96
Percentage of results with fitness >0.98	20	35	50	35
Mean time to success (generations)	243	390	288	271

Notes. The statistics are calculated from the results of 20 runs for each configuration.

Following these results, a further 20 evolutionary runs were executed with the configuration of controller A, selected because it gives the highest-dimensional search space without using feedback. While in many search algorithms increasing dimensionality is a problem, in evolutionary algorithms it can be advantageous in that it may introduce more pathways to success [5]. Runs continued until the search could be seen to have either succeeded or effectively halted at a local optimum, up to a maximum of 1000 generations. In this case only four runs met the success criterion, but the average fitness was high, as can be seen in Table 3.

Sets of evolutionary runs with configurations based on that of controller A were later conducted in an attempt to improve upon the evolutionary success rate of configuration A (Table 3). The first set of 20 was with a configuration, referred to as J, which had the input weight range doubled to $[-4, 4]$. This change was based on an observation that controllers that just failed to succeed always did so because they did not respond correctly to the most distant falling objects, which are initially detected only by the outermost sensors, and only briefly so if the correct response is not immediately produced. Adjusting the range of sensory weights seems to have made it possible for agents to respond more strongly, and correctly, in these most difficult trials. The configuration of the second set of 20, referred to as S, was based on configuration J, but in this case spring parameters were not evolved and spring velocities were not used in the readout. This process is more faithful to the reservoir computing paradigm, and was surprisingly successful given the low dimensionality for controller tuning. The final set of 20, referred to as W, was also based on configuration J, but was modified so that instead of each sensory input driving only a single node in each MSD network, each free node received a weighted input from every sensory neuron.

The boxplot in Figure 3 shows the distribution of fitness scores found for the four sets of evolutionary runs. Configurations J, S, and W are all more successful than A in terms of mean fitness. This plot by itself makes J and W look strongest, but when we look at other statistics, such as the number of runs that succeed according to the 98% criterion, and how long it takes to obtain those successes (Table 3), a slightly different picture is painted. Configuration S, perhaps because it has a

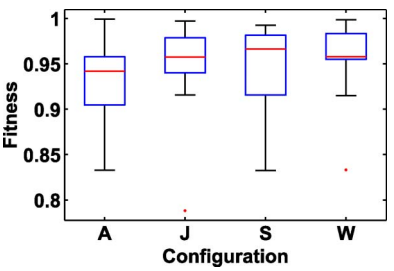

Figure 3. Fitness distributions for controllers with configurations A, J, S, and W, over 20 evolutionary runs for each configuration.

far smaller search space than all others, has a 50% success rate, considerably the best of the four. The reader will be reminded that the proportion of the population replaced in every generation varies, so not much can be drawn from the mean time to success for all configurations, but they seem comparable. It may be that a further improvement may be made by merging configurations S and W, which will still give a relatively small search space, although with the advantage of a number of weights to be tuned for each input. However, we have yet to see how the various configurations cope with evolution in noisy conditions, which will be the ultimate test. We have also yet to conduct a proper investigation into the best evolutionary algorithm for this task.

In summary, a number of different configurations of MSD networks and network layouts (Figure 4) have proven effective. Interestingly, there is also considerable variety in the behavior of successful controllers, and, from a high-level point of view, various strategies are possible. Broadly speaking, three distinct strategies have been observed. In Figure 5 we show an example of each. The individual graphs plot the horizontal distance between agent and object over time. When the agent's behavior is viewed from this perspective, it can be seen that controller A inches towards objects until it can distinguish between them, controller A5 finds objects quickly and then oscillates around their position until making a decision, and controller A1 scans back and forth.

Understanding how these different behaviors arise will help us to understand how compliant networks can generate adaptive behavior. The following section will thus analyze the behavior of MSD networks in this task in general, and finally we will analyze two specific controllers in detail in Section 5.

4 Analysis

In order to uncover the principles of operation of successful MSD networks, we conducted a number of analyses. We test the hypothesis that controllers actively maximize information from the sensors. We examine the capacity of the networks for memory in order to determine if it is possible that the agent can integrate the inputs from its sensors over time, and we disrupt the normal operation of networks in a variety of ways in order to gain some knowledge from the effects.

In all of the following analyses, except that of perturbation analysis, we will examine two results, for comparison and contrast. The first is controller A, from our first evolutionary run, and is selected because its behavior is characteristic of the results as a whole. The second, denoted A1, is a result from an evolutionary run with configuration A and is selected because it is unique: It is the only

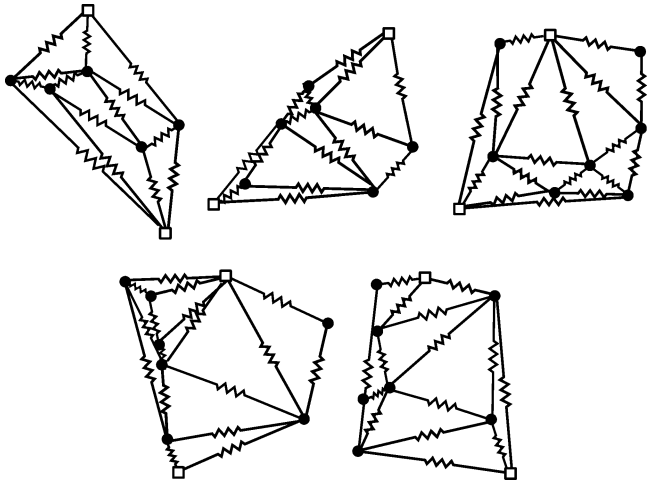

Figure 4. Topologies of the first successful evolved controllers. From left to right, the top row shows controllers A, B, and C, and the bottom row shows controllers D and E. (Dampers and some springs and nodes are omitted for clarity.)

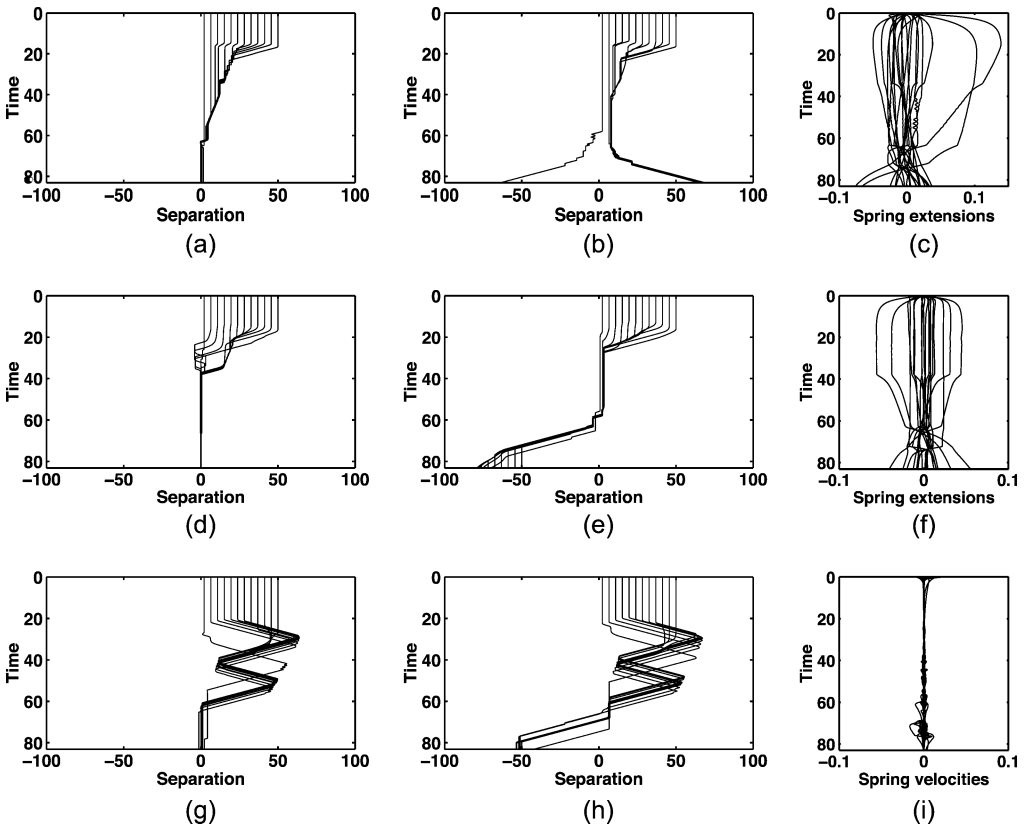

Figure 5. (a), (b), (d), (e), (g), (h): agent trajectories throughout all evaluation trials. Trajectories from catch trials are shown on the left, and those from avoid trials on the right. Trajectories from three controllers are shown: from top to bottom, controllers A, A5, and Al. In all cases the behavior may be split into two phases, the first where the agent moves to keep the object in its visual field, and the second where it catches or avoids appropriately. (c) Spring extensions from one network of controller A in trial 7 (catch). (f) Spring extensions from one network of controller A5 in trial 5 (catch). (c) and (f) both show examples of energy stored in the networks throughout the trial. (i) Spring extension velocities from one network of controller Al in trial 19 (avoid). Note that the agent is in motion for most of the trial, as seen in (h), but the spring velocities are very close to zero.

successful MSD controller that we have yet seen that scans objects in a similar fashion to Beer's CTRNN [3, 4].

Before continuing to more detailed scrutiny of individual controllers, certain observations can be made regarding the problem and the results in general.

In each trial there is an initial period where the falling object is out of range of the agent's sensors. Because sensory input is normalized and scaled, in this period the sensory neurons receive a constant input of −0.5. Thus the MSD networks are driven into a biased configuration, which will affect the response to stimuli from the object when it falls into range. The MSD networks have nonzero output throughout this period, but due to the symmetry of the network pair and the antagonistic motor configuration, the agent does not move until the object is detected.

In general, as the objects fall towards the agent, the sensory inputs to the networks rise along a ramp from −0.5 towards 0.5 (see, for example, Figure 14c). The initial biased configuration takes the form of energy stored in the networks, and so, as the inputs rise to zero and beyond into positive values, energy is released from the networks and then imparted to them again. However, as some sensors are never activated or are activated only at certain times, there will always be some energy stored in the networks. Figure 5c gives an example of this. Both extended and compressed springs

indicate stored energy, and in this case the total energy rises quickly, then falls gradually as the sensory stimuli approach zero, and then rises again close to the end.

The behavior of the agent over the remainder of the trial can be split into two phases (Figure 5). In the first phase the agent positions itself under the object, not necessarily centrally or statically, as we shall see in Section 5, and in the second phase the agent responds to the type of the object in either catch or avoid actions.

A point worth noting is that the two objects are of different widths (Figure 1). Circular objects are 30 units wide, but although diamonds are squares 30 units wide, they are so oriented that the agent sees the length of their diagonal, which is approximately 42. This means that if the agent sits below the object, at some point more sensors will be activated for diamonds than for circles, something that may be exploited by evolution.

Finally, while it is not impossible that a network with fading memory could exploit some transient effect as a temporary timer, we do not consider it likely that networks as small as those herein, with a high degree of coupling throughout, could isolate such an effect over the period of the trial. However, as already noted, for all successful agents the sensory inputs rise along a ramp. Hence, while the pattern of sensors that are actually activated may vary across trials and across agents, there will still be a strong correlation between the magnitude of stimuli and the present point in time, a feature that is ripe for exploitation by the evolutionary process in producing successful behavior.

4.1 Information Analysis

In this section we consider an information-theoretic explanation for the behavior of the agent. Mobile adaptive agents do not only process information from their sensors, they also move to generate it. In this case all successful controllers moved the agent so that the falling object was kept in its visual field until the decisive moment of selecting the appropriate action for the object type. This is of course an obvious strategy, but when the visual field is examined through an information-theoretic lens, it becomes evident that some controllers have evolved to achieve this by maximizing sensory information, for at least some of the time. In the following we explore how far this is the case, and over what time scale this adaptation takes place.

For each trial we subject our agents to, what the agent will see at any particular point in space and time is predetermined, as the horizontal position and vertical velocity of the object are constant, unaffected by the agent, and their apprehension is unaffected by noise. The lookup tables we used for the sensor model essentially extend over the entire space and time of the trial, and so can be easily analyzed for informational content. We quantized the stimulus in the tables to 200 distinct levels. For each location in space-time we treat the combination of seven stimuli as a single event, and then calculate the probability of occurrence, over all space-time, for each event. From this probability mass function we then compute the self-information, I, of each location (x, t), using the following equation:

$$I\big((x,t)_k\big) = -\log_2(p_k) \tag{12}$$

where p_k is the probability of the pattern of stimuli at point k.

In this context, patterns of stimuli that have low probability have high informational content, because they have low ambiguity as to the position of the agent relative to the object. The self-information of the agent's visual field, as a function of its horizontal distance from the object and simulation time step, can then be viewed as a kind of map across space and time (see Figure 6a and b), where the ambiguity of the pattern of stimuli falls as the self-information rises.[1] The maps we show here were built with the assumption that the agent could not know the current time. As noted

1 To clarify, although entropy is the expected value of self-information, we do not consider the entropy of the space-time map, as its structure is fixed. In addition, we do not consider the entropy of the signals from the sensors over time, as it is not guaranteed that this entropy will be correlated with paths of high information through space-time.

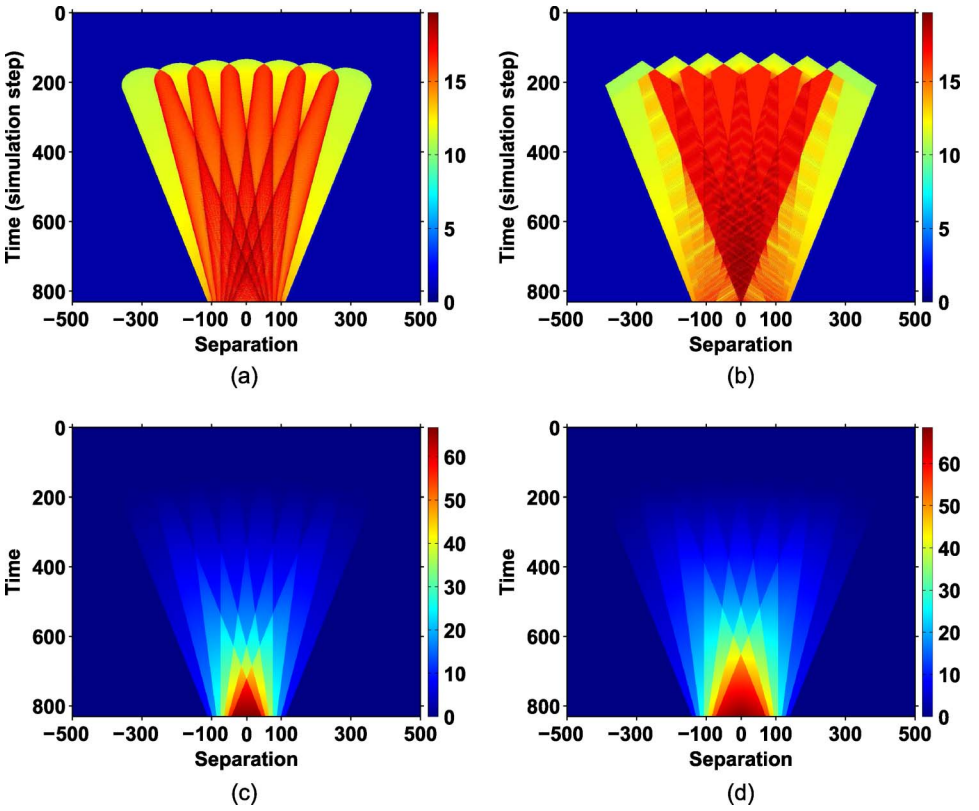

Figure 6. Representations of the visual field of the agent. As the object falls at a constant rate over all trials, there is an equivalence between its position and time, plotted on the *y*-axis. The horizontal distance between agent and object is plotted on the *x*-axis. The same cells, bounded by edges where sensors switch on or off, can be seen in both representations. (a) The self-information of the visual field for catch trials. (b) The self-information of the visual field for avoid trials. (c) The summed intensity of stimuli over the visual field for catch trials. (d) The summed intensity of stimuli over the visual field for avoid trials.

in Section 4, there is an indication of simulation time available in the environment due to the objects falling, so as a control we also built maps based upon the opposite assumption: that the agent knew exactly what time it was. In this case, rather than having a single probability mass function over all time, we constructed one for each simulation step and so calculated the self-information instant by instant. The resultant maps had roughly the same structure as the ones we show here, and so for all following analyses we use only the maps based on the first assumption.

Beer [4] identified a structure in the visual field of the agent. Cells of continuously varying levels of sensory stimulation are divided by *edges*, which delineate the boundaries between individual sensors detecting and not detecting the object, as the relative position of object and agent is varied (Figure 6). When we converted the visual fields for both catch and avoid into maps of the self-information of sensory events over space-time, we found that the informational structure has a certain amount of correspondence with the already noted structure, shown in Figure 6c and d, but also that the aforementioned edges have particularly high informational content (Figure 6a and b). Furthermore, when we overlaid the trajectories of agents onto these maps, we saw that some controllers, for catch trials in particular, appeared to closely follow these edges (Figure 7). As noted in Section 4, the behavior of agents can generally be split into two phases: moving to get a good view of the object, and then moving to catch or avoid as appropriate once the object type is identified. Visual inspection of the trajectories across these maps suggests that agents may be following a route of high information in phase 1, but it is not clear if this is also the case in phase 2.

The analysis reported here is related to *empowerment*, an information-theoretic measure introduced by Klyubin et al. [17], which essentially quantifies the effect of an agent's action upon its own apprehension of its environment. They define empowerment as the channel capacity between the action of an agent at time t and the state of its sensors at time $t + 1$. The underlying idea is that it is in the agent's interests to maximize this informational return from its actions, and that this "empowers" it. Trajectories such as that for controller A on catch trials (Figure 7a) suggest that the evolutionary process has identified and exploited paths of high information, but it is unclear to what extent. It is also unclear whether the evolved controllers have any capacity for the detection and measurement of information, or whether they simply follow evolutionarily predetermined routes.

We are able to shed some light on this subject by considering the extreme case: an agent with the policy of, at each simulation step, moving to the position of highest information in its reach. As we can see in Figure 8a and b, this is an effective policy for phase 1, but is not sufficient for phase 2, when the object must be identified and responded to appropriately. Therefore we can rule out the simple policy of always maximizing information at the evolutionary time scale and at the level of individual controllers. In fact a policy of simply maximizing the summed amplitudes of stimuli compares well in performance with maximizing information (Figure 8c and d). This is also the case when noise is added to the information map or the visual field for the two scenarios, respectively (Figure 9). Although we have not yet identified it in any controllers, we believe that in general it will

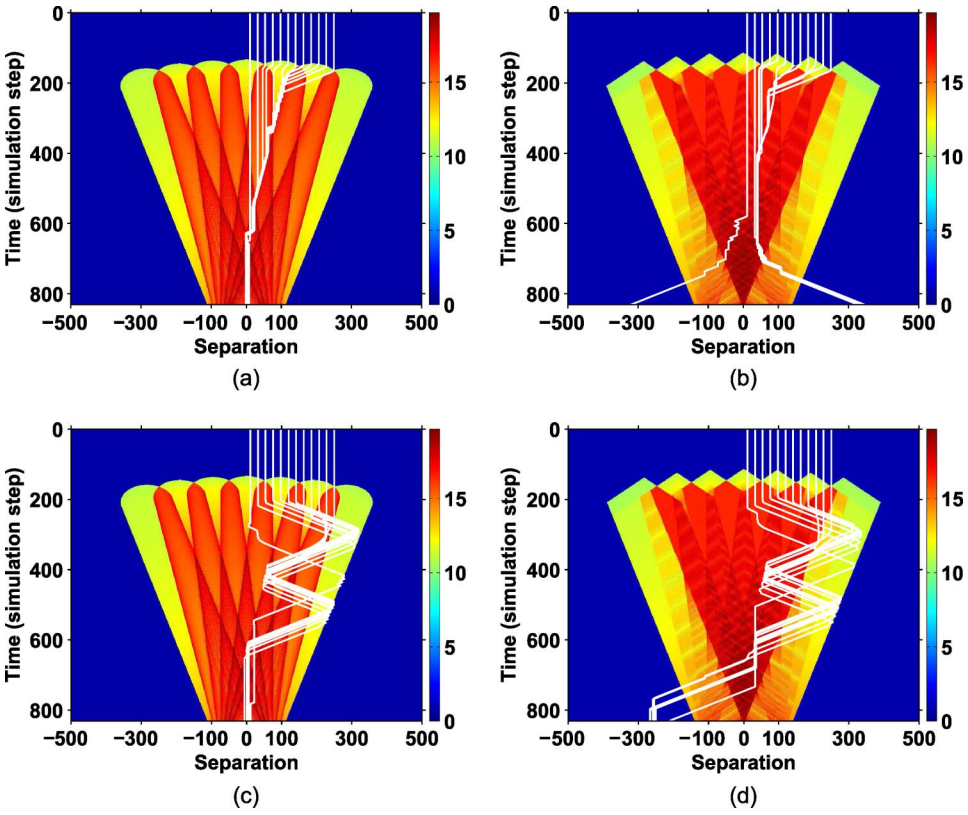

Figure 7. Trajectories for controllers A and A1 overlaid on the self-information maps. (a) The trajectories of controller A for catch trials overlaid upon the self-information map. (b) The trajectories of controller A for avoid trials overlaid upon the self-information map. (c) The trajectories of controller A1 for catch trials overlaid upon the self-information map. (d) The trajectories of controller A1 for avoid trials overlaid upon the self-information map. Controller A clearly follows an edge of high information in catch trials, and controller A1 appears to use the outside edge where information suddenly drops to almost zero to decide when to turn back towards the object during its scanning motion.

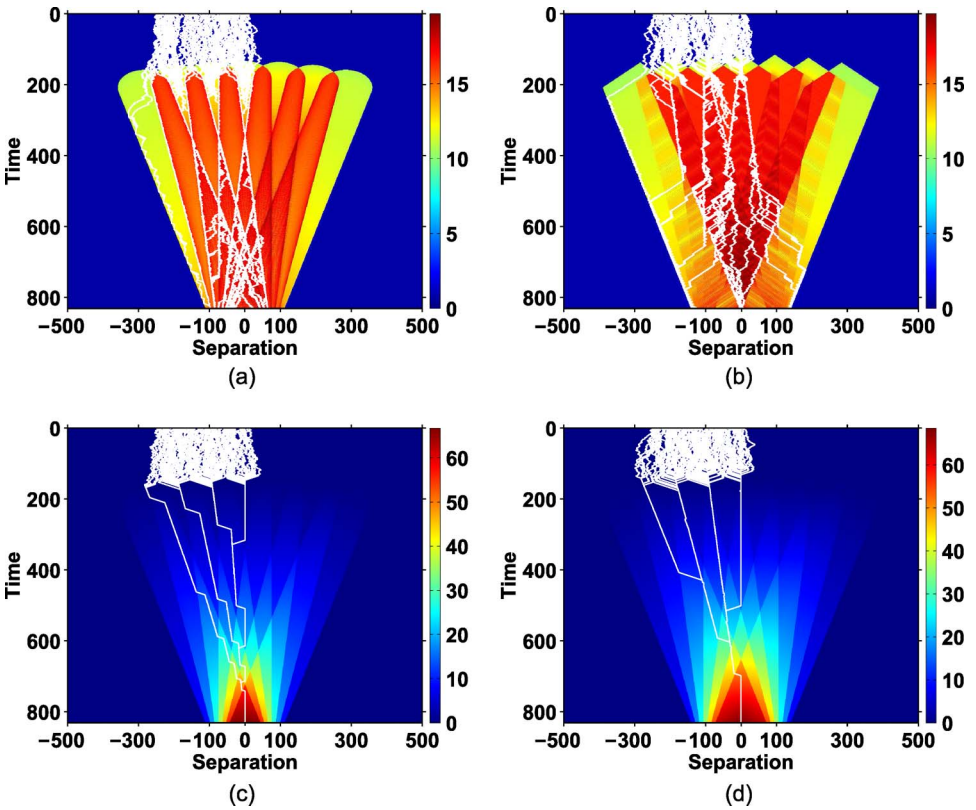

Figure 8. Testing simple agent policies. The first policy, shown in (a) and (b), is for the agent to move so as to maximize the self-information appearing to its sensors. The second policy, shown in (c) and (d), is for the agent to move so as to maximize the sum of stimulus intensities appearing to its sensors. In both cases, 100 trials are shown, starting from random positions in the same interval as the trials used in evolution. (a) Maximizing self-information for catch trials. (b) Maximizing self-information for avoid trials. (c) Maximizing summed stimulus intensity for catch trials. (d) Maximizing summed stimulus intensity for avoid trials. Note that maximizing stimulus intensity is optimal for catching objects, but shows no distinction between object types. On the other hand, maximizing information, while suboptimal for both catch and avoid, does lead to distinct behaviors.

be easier for both evolution and individual agents to follow this second policy, and also that it will be more robust at the evolutionary time scale. For example, in preliminary tests with noisy sensors it appeared that controller A, which has clearly evolved to follow a path of high information in phase 1, was too tuned to specific patterns of stimuli and failed dramatically with even low noise levels.

In summary, while evolution has exploited the structure of the information space, the policy of simply maximizing information suffers from a lack of distinction between the quantity of information and its value. While we may expect a certain correspondence between the quantity and quality of information in biological organisms, where sensory systems are coevolved with behavior, this should not be taken for granted. In systems such as this, where the sensory morphology is hand-designed and fixed throughout evolution, it is even more likely that a certain amount of received sensory input will be redundant. As we shall see in Section 4.4, performance is more impacted upon by certain sensory disruptions than by others, which lends further support to this idea.

4.2 Capacity for Memory

We turn next to measuring the capacity for memory of networks A and A1. As with the analysis of the preceding section, this question is of interest in that it gives clues as to how the controllers

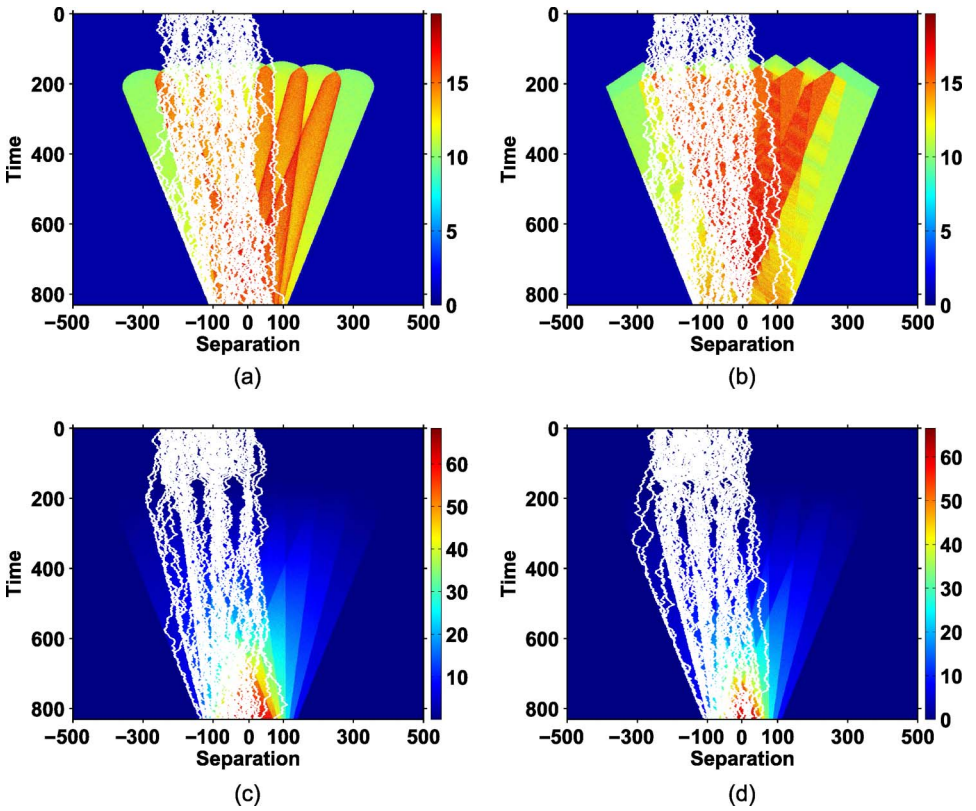

Figure 9. The policies of maximizing self-information and maximizing summed intensity of stimuli are tested under conditions of noisy sensors. The visual fields are subjected to white noise, with a signal-to-noise ratio of 5 : 1. (a) Maximizing self-information for catch trials under noisy conditions. (b) Maximizing self-information for avoid trials under noisy conditions. (c) Maximizing summed stimulus intensity for catch trials under noisy conditions. (d) Maximizing summed stimulus intensity for avoid trials under noisy conditions. In this case both strategies are still successful in maintaining attention on the object in phase 1, but lead to no distinction between object types.

perform their functions. In acquiring our measures, we follow a method based on that described by Maass et al. [22]. They 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 sensory 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: the sum of their intensities. In this case many segments of the input stream have high correlation with one another because, due to the tendency of agents to position themselves under an object until it can be recognized, the general trend is for sensory input to increase along 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, 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 a capacity for memory in the network. The same test is performed for readouts that receive only spring extensions as inputs, readouts that receive only spring velocities, and readouts that receive both. Previously we

conducted this test with a small set of consecutive segments of the input and output streams [16]. Here we modify this approach so that the compared segments are still of the same length and relationship to one another in time, but we compare all possible segment pairs over the course of a trial. The segment lengths are 10 s, or 100 simulation steps, long, and the output segment starts where the input segment ends. The two segments are then moved as a pair of sliding windows across the entire time series and the relevant correlations are calculated at each point.

The results of some of these tests are shown in Figure 10. 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. In Figure 10a and b we see that controller A shows some signs of memory capacity, although in general the baseline is already very high. For the avoid trial (Figure 10b) there is a period, starting from around step 550, 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. While a negative correlation still indicates a relationship, for the sign of the correlation to change introduces ambiguity, which the trained readout has been able to resolve. However, it appears unlikely that this controller exploits memory capacity. As shown in Figure 5a and b, for both catch and avoid behaviors, initially this controller gradually creeps towards the object as it falls, suggestive of a purely reactive network. The result for controller A1 also indicates a degree of memory capacity, with the network being able to recover more information about earlier input segments than the input stream itself, as shown in Figure 10c and d, and again to resolve the ambiguity we see in the baseline. This is consistent with the general strategy. Unlike all other results, this controller drives away from the object and then returns to it, a behavior that seems of a

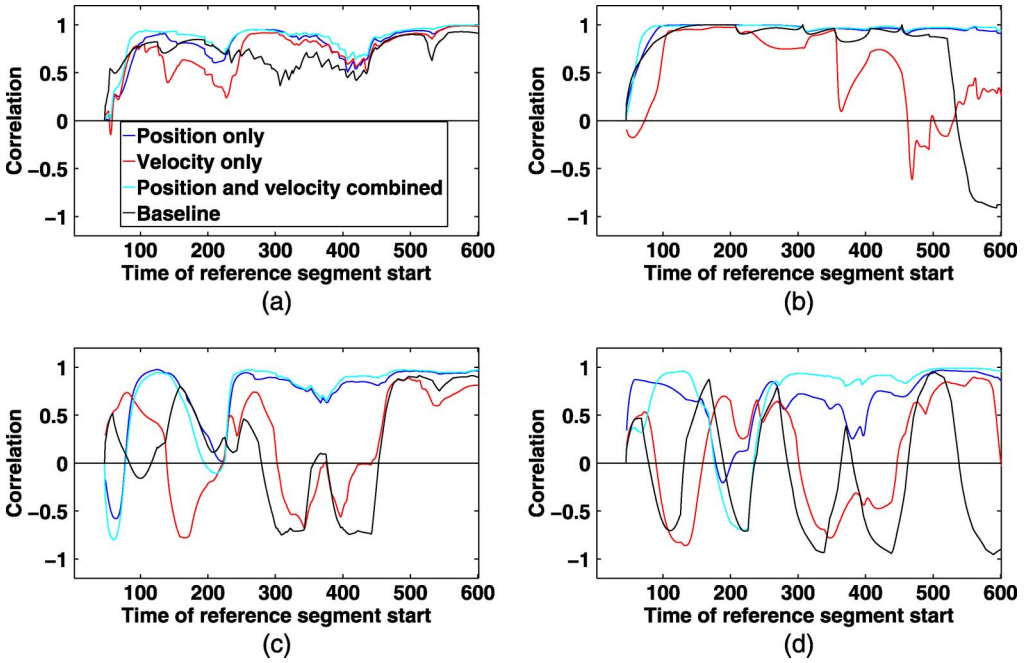

Figure 10. Measuring memory capacity. The linear readout is trained to recover inputs from 10 s ago. Then the input stream and the controller output are split into 10-s segments, and the correlation of each output segment with the prior input segment is calculated at all available points in time. The correlation of the input streams from the two periods is also shown as a baseline. The legend for all four plots is shown in (a). (a) Controller A, trial 5 (catch). (b) Controller A, trial 18 (avoid). (c) Controller AI, trial 5 (catch). (d) Controller AI, trial 18 (avoid). In all plots the baseline trace indicates the zero-memory case, and the memory is the difference between network outputs and this baseline. Typically the baseline is high and the network memory is therefore small, but it should be noted that both networks successfully disambiguate the case when the baseline correlation switches sign and successfully track the sign of past inputs.

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.

4.3 Perturbations

We can learn something more about these controllers and the difficulty of the search problem by examining the effect on performance of adjusting the evolved parameters. This analysis has only been conducted for controller A1. Parameters are perturbed by adjusting genes and then mapping to phenotypes to evaluate performance. We adjusted genes, one at a time, with a two-part scheme: with fine increments near to the existing value, and larger increments covering the entire possible range.

In our first test (see Figure 11) we used an increment of 0.2 across the whole range, and 0.1 in the range of [−0.2, 0.2] centered about the value already optimized by evolution. At this scale we see a fairly smooth fitness landscape, with few local optima, although as far as only adjusting a single gene at a time goes, there seems to be no possibility of improvement upon this result, from this location in parameter space. However, that does not rule out the possibility of improvement by adjusting multiple parameters simultaneously.

At this scale certain characteristics of controller A1 become clear. When varied individually, the nonlinear spring coefficients, encoded in genes 80 to 115, have very little bearing on the performance of the agent, with fitness remaining high throughout. This seems to be because the extensions and velocities of the springs are very low (Figure 5c, f, and i). As can be seen in Equation 7, the nonlinear effects are proportional to the cubes of these states. As the states are already well below zero, cubing them leads to only micro effects. It is worth mentioning here that this task may not even require nonlinearity in the network, although, as noted by Hauser et al. [12], the couplings between the MSDs in the network will introduce a certain amount of nonlinearity regardless of which spring model is used.

One discovery that is slightly more surprising is that the genes that encode for the weights applied to the velocities of springs in the linear readout, genes 19 to 36, also have a relatively low effect on fitness. A possible reason for this is that in general the inputs to the network increase along a ramp. The MSD elements will typically have highest velocity shortly after sensors are switched on or off, which can provoke a sharp change followed by a transient vibration; but gradual change at the inputs also drives gradual (i.e., low-velocity) change in the network. As exampled in Figure 5i, the velocity states will, on average, be low, and so the weights applied to them will be of small importance. On the other hand, in our tests for memory capacity in this controller, we have seen that velocity encoding is advantageous in recovering memory effects. The other side of this coin is that

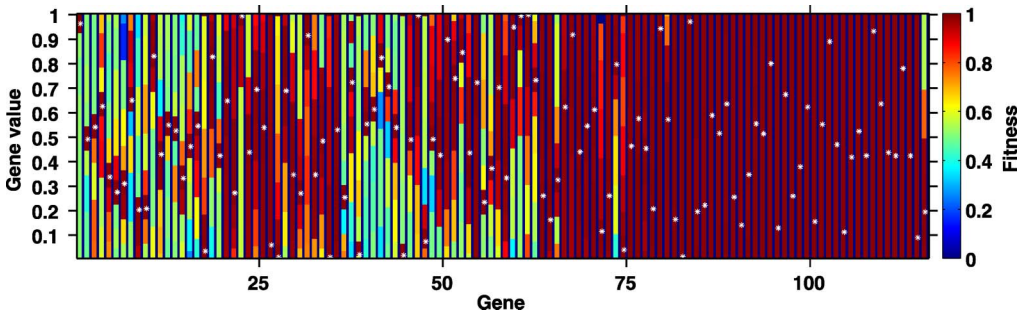

Figure 11. Each gene was perturbed one at a time for controller A1. The entire range of values for each gene is explored with resolution 0.2, and the interval of [−0.2, 2] about the evolved gene value is explored with resolution 0.1. Evolved gene values are marked with white asterisks. For many genes the surface contains one or more plateaus, indicating large regions with no cues for evolution to determine which way to move parameters towards their optima. The values of many genes, particularly on the right-hand side of this plot, can be seen to have very little effect on fitness.

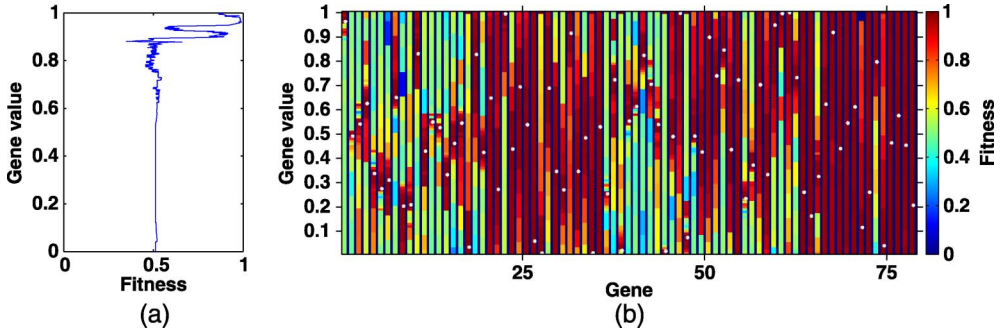

Figure 12. (a) The one-dimensional fitness landscape for gene 1 of controller A1, where all other genes are held at the evolved values. Here it becomes clear that not only are plateaus in the fitness landscape a problem, but the surface is increasingly noisy in the region surrounding the optimum value. (b) Each gene from 1 to 79 was perturbed one at a time for controller A1. The entire range of values for each gene is explored with a resolution of 0.1, and the interval of [−0.1, 0.1] about the evolved gene value is explored with resolution 0.001. Evolved gene values are marked with white asterisks.

incorrect velocity encoding will be disruptive of memory effects appearing at the network outputs, and indeed of behavior in general. At this point we can only speculate, but it may be that although we see little effect when we perturb these weights individually, the effect of adjusting several simultaneously will be deleterious.

Also unexpected is the discovery that genes 62 to 79, which encode the MSD linear damping coefficients, may in most cases also be varied without too much effect. This is surprising because the character of the response of an individual MSD can vary dramatically with the damping coefficient. We believe the relative neutrality of individual damping coefficients is due to two causes: Firstly, the damping effect is proportional to the velocity of the spring, which we have already seen is often close to zero. Secondly, and more importantly, due to the coupling of MSDs through nodes, damping may be a property that is local more to a node than to a single MSD. Therefore the effect of changing the damping coefficient for a single element may be partly absorbed by adjoining elements.

When we repeated the test using an increment of 0.1 across the entire gene range and an increment of 0.01 in the range of [−0.1, 0.1] centered about the existing value, we discovered that the fitness landscape is in fact not as smooth as it originally appeared (see Figure 12b). We performed this test for genes 1 to 79 only, as we had already observed that genes from 80 upwards were not critical. Finally we varied gene 1 over the full possible range, at a resolution of 0.001 (see Figure 12a). This one-dimensional fitness landscape is extremely difficult to negotiate, with a large plateau over much of its range and a highly disrupted surface in the region of the global optimum. It appears that the results given here are fairly characteristic, and that this explains the difficulty of obtaining successful controllers.

4.4 Lesions

As with perturbations to the system parameters, we have learnt something about out controllers by recording changes in agent performance as parts of the network are disabled. Four experiments, illustrated in Figure 13 for controllers A and A1, 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 experiment, one sensor at a time was disabled and 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 set their nonlinear coefficients to zero.

Various observations can be made from these plots. It can be seen that in general adjustments to the network elements for controller A (Figure 13a–c) tend to cause failure in catching circles far

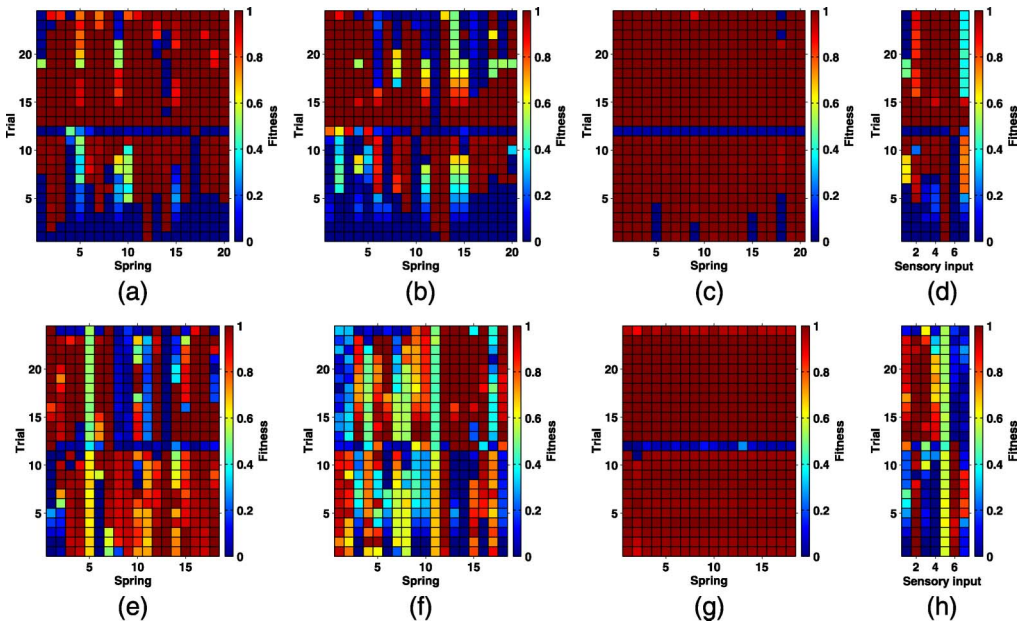

Figure 13. Lesion experiments. The top row of plots shows results for controller A, and the bottom for controller A1. The rows in plots show the performance on a trial-by-trial basis. The color of a grid element shows the fitness of the controller following the lesion. Trials 1 to 12 are the catch trials, and trials 13 to 24 are the 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 linearized; one input at a time is disconnected. It can be seen, here as in the perturbation tests, that the performance for both networks is relatively robust to the adjustment of many parameters.

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, but depends on all sensors except sensor 5 for catching (Figure 13d).

Complete removal of springs from the network causes high failure in both agents, although controller A1 (Figure 13e–h) is more robust to this than controller A. That this should cause a high failure rate comes as no surprise, given the tightly coupled nature of the network dynamics. Neither controller shows a strong dependence on spring nonlinearity; that is in accord with observations in the previous section on perturbation tests for controller A1.

The plots in Figure 13 suggest that for both of these agents the most difficult trial is trial 12, the last where catching behavior 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 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.

5 Case Studies

In this section we will consider the controllers A and A1 separately, scrutinizing their behavior over representative trials in some detail. What we can observe at this level will be supplemented by the observations of the previous analyses, and so we will build as complete an understanding of these two controllers as we are yet able to. As these two seem to represent the opposite extremes of the range of behaviors we have seen in our collection of results, the two together may be considered broadly representative of the class of controllers under the evolutionary conditions we employed.

5.1 Controller A1

As can be seen in Figure 14b, agents tend to operate their motors in the regions of cutoff and saturation, a consequence of using a sigmoid operation between network output and motor (Equation 2). Figure 14a shows the network outputs that produced the velocity shown in Figure 14b. The two red lines show the point at which individual motors switch off and saturate. Because the networks drive an antagonistic motor pair, the agent stops at every point where the network outputs coincide. It also stops whenever the motors are either both saturated or both switched off. From the point of view of dynamics the network outputs appear complex and nonlinear, especially as sensory input tends to rise linearly and show high correlation not only between past and present but also between different sensors. From the point of view of information, the sigmoid in the motor function operates as a filter, rejecting redundant information from the network outputs.

In the case of this controller, it can be seen that throughout most of the trial there is a certain degree of symmetry between the network outputs. Spikes tend to indicate short-term effects of events in the sensory streams, whereas divergences and convergences occur relatively periodically (suggesting a certain indifference to many stimuli) until close to the end of the trial. In tests with white noise added to sensory input, the controller represented in these plots showed the highest degree of robustness to white noise yet seen.

It is often difficult to attribute causality in the sensorimotor loop. For this controller, the trajectories across the information maps shown in Figure 7c and d suggest that this controller is averse to a lack of information, turning back towards the object at the point where the last sensor switches off. But if we look at $t \approx 300$ in Figure 14b and c, where the agent makes such a turn, we discover that the last sensor

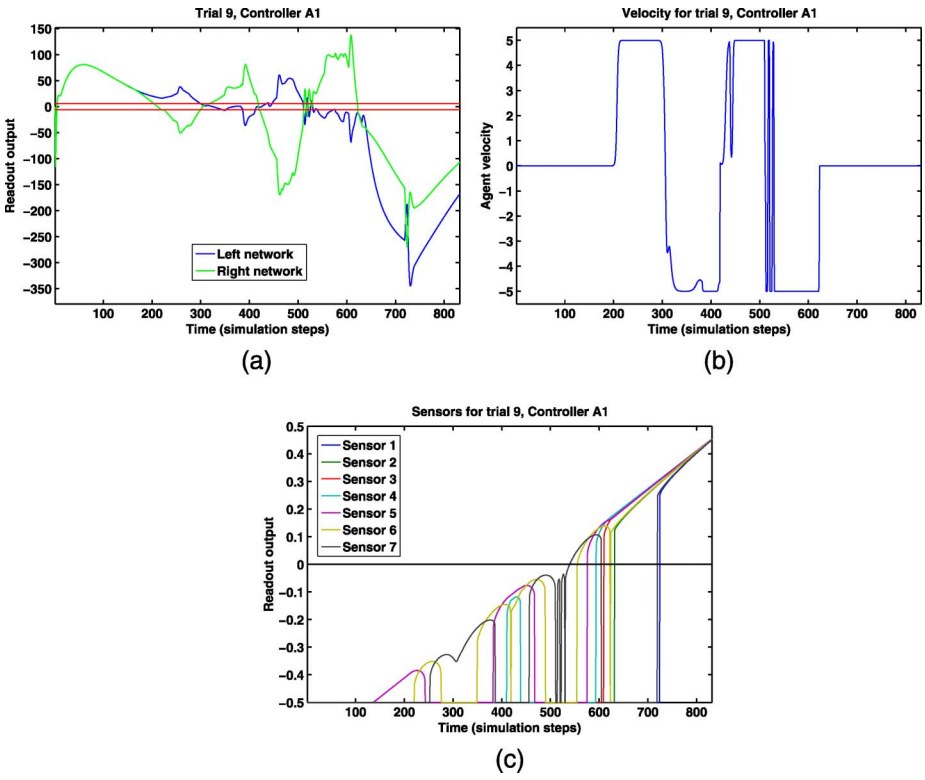

Figure 14. Network outputs, agent velocity, and sensory stimuli of controller A1 over trial 9 (catch). (a) Network outputs. (b) Agent velocity. (c) Sensory stimuli. Note that the sensory stream here and in following figures is recorded after being normalized and shifted to the interval [–0.5, 0.5], but prior to the delaying effect of the sensory neurons.

in fact did not go off for trial 9. We can also see that there is no easily discernible effect of the previous sensor switching off. Therefore we conclude that, for this sensor on this trial and at at least some points, the causal relationship between sensor and agent velocity is the reverse of our expectation—the sensor output is determined by an inevitable turn rather than being its trigger.

It is still of interest why evolution has selected a controller that produces a trajectory that appears to respond to edges in the information map by changing direction (Figure 7c and d). Perhaps the ancestors of this controller responded to those edges, but at some point the relationship changed from reaction to prediction. Another striking pattern in the trajectories is the apparent constancy of the magnitude of the gradient throughout the period where the agent is scanning, caused by the fact that in this period, while the agent changes direction a number of times, its magnitude of velocity is generally at the maximum possible. Another question to be asked here is this: Given a lack of fast reactions to sensors and an apparent scanning behavior, how far does the network integrate sensory input? This is not an easy question to answer—generally speaking, the sensory input is constantly growing for any successful agent—this is a kind of environmental integration that can potentially be exploited by evolution. It is unclear how far this controller exploits sensory input simply as an energy source to drive its oscillation, and how far it is responding to particular sensory patterns.

However, the controller is capable of distinguishing between the two objects, so it is clear that at least at some point the sensory input is more closely observed. The point of decision would appear to be soon after $t \approx 600$, where the network outputs diverge in a way that shows no symmetry. In trial 9, which is a catch trial, the motors switch off at this time and the agent stops to perform a catch. However, as can be seen in Figure 15a and b, for the corresponding avoid trial, there is some

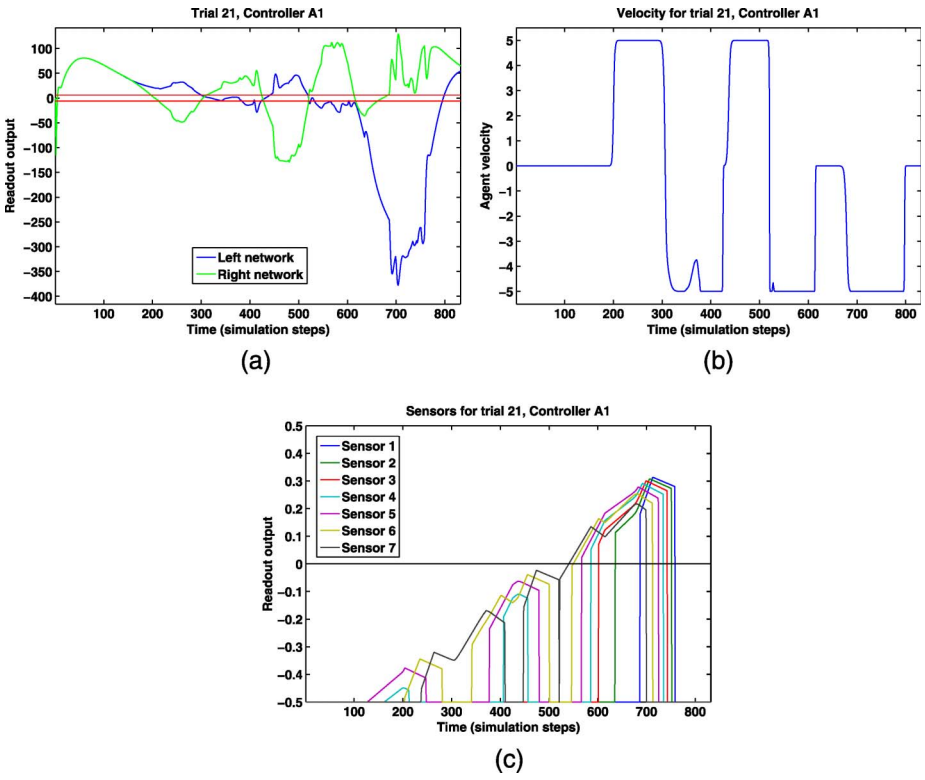

Figure 15. Network outputs, agent velocity, and sensory stimuli of controller A1 over trial 21 (avoid). Note that in this trial the object falls from the same position as in trial 9 (Figure 14). (a) Network outputs. (b) Agent velocity. (c) Sensory stimuli.

delay between decision and action. The network outputs diverge soon after $t \approx 600$, but as both networks are well within the cutoff region at that time, the effect is delayed and it takes until $t \approx 670$ for the right motor to switch on (Figure 15b) and the escape behavior to be initiated.

5.2 Controller A

Whereas controller A1 tends to switch on both motors, the general pattern for controller A is to keep the left motor at full power until late in the trial and so control its velocity by switching the right motor on and off (Figure 16b). In catch trials it inches towards the object by pulsing the right motor. In avoid trials it is still the right motor that effectively controls the velocity, but its action can be smoother and more prolonged than in catch trials (Figure 17b).

In general sensory effects on network outputs are easier to detect for this controller than for A1. The first sensor to activate for the catch trial number 7, sensor 3 (see Figure 16c), appears to provide the energy for the left motor, with the network output tracking the ramp of that input and therefore staying well in saturation until the decisive moment. For this network the effects of other sensors are superimposed upon that ramp in the output, but in the right network there appears to be little or no effect of sensor 3, with the underlying form of the output in phase 1 being initially flat and not far into saturation. There is a distinctive curve superimposed upon the effect of sensor 3 in both networks; this is the effect of sensory stream 4.

In this trial sensor 2 has a repulsive effect on the agent throughout phase 1. Each time it comes on, it is followed by a negative spike in the right network, which briefly deactivates the right motor

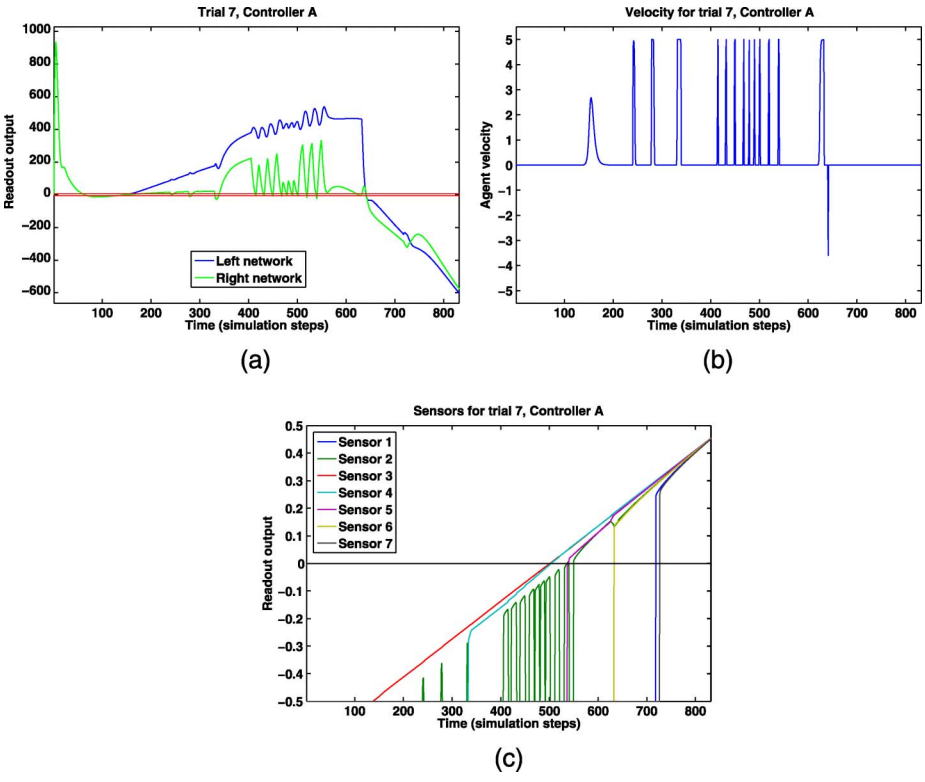

Figure 16. Network outputs, agent velocity, and sensory stimuli of controller A over trial 7 (catch). (a) Network outputs. (b) Agent velocity. Note that, in contrast to controller A1, this controller tends to pulse motors. (c) Sensory stimuli. Note that the sensory ramp for this controller has a remarkably constant gradient compared to other controllers. In phase 1 this gradient corresponds to an edge of high information in the visual field (Figure 7a).

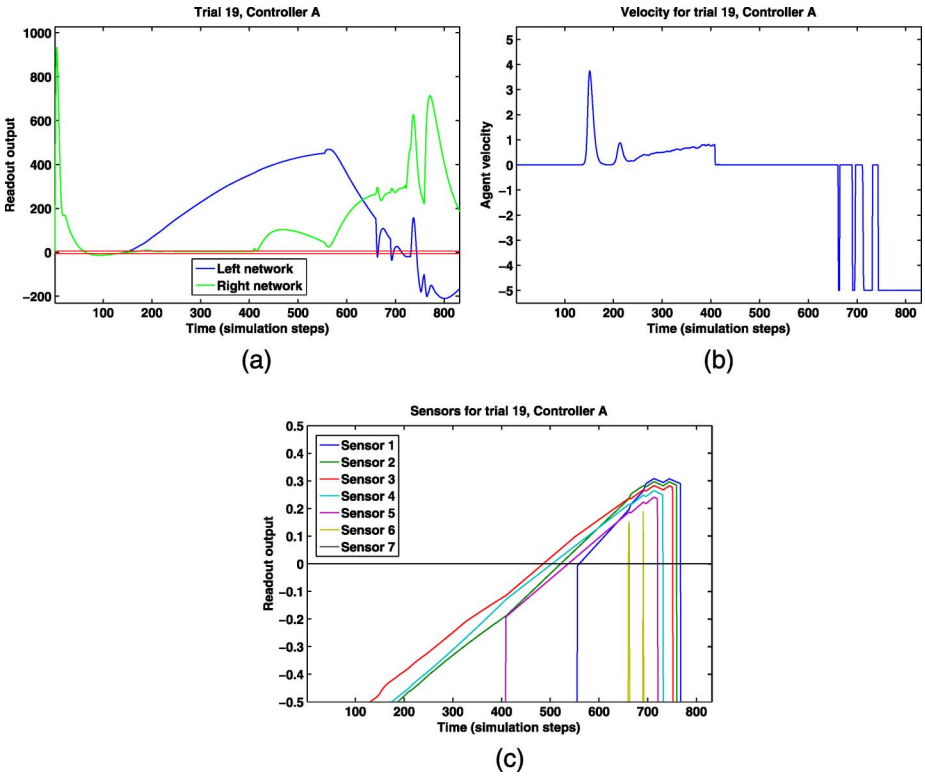

Figure 17. Network outputs, agent velocity, and sensory stimuli of controller A over trial 19 (avoid). Note that in this trial the object falls from the same position as in trial 7 (Figure 16). (a) Network outputs. (b) Agent velocity. Note that although there is still a tendency to pulse motors, in the avoid trials the velocity occasionally varies more smoothly, as can be seen here between $t \approx 200$ and $t \approx 400$. (c) Sensory stimuli.

so that the left motor moves the agent until the sensor also switches off. It appears that the activation of sensor 5 at $t \approx 540$ changes the predisposition to move away when sensor 2 is activated. The decisive moment in catching occurs when sensor 6 switches on, at around $t \approx 633$; the sensory input balances and the two network outputs rapidly converge (see Figure 16c and b for sensors and velocity, respectively, over this trial).

The main points to be made here are that sensory effects on the network outputs are almost always immediate for this controller, and that their effects are precariously balanced. In the description above we have emphasized the importance of sensors 2 and 6 in centering and catching, respectively, and yet in the lesion tests of Section 4.4 it emerged that in the case of trial number 7, the most detrimental lesions were for sensors 3, 4, and 5 (Figure 13d). A closer look at Figure 16a and c reveals that although sensor 2 seems to drive motion, it is the combination of its effect and that of sensor 4 that ensures that each time sensor 2 switches on, the velocity response is only a short pulse. This intricate balance of sensory input corresponds to the path of high information that this agent was seen to follow in Section 4.1 and has been discovered not to be a robust solution.

In trial 19, the corresponding avoid trial to trial 7, the response to individual sensors appears somewhat different. Presumably because sensors 2 and 4 are activated almost simultaneously, the effect of sensor 2 is somewhat muted, with a low velocity moving the agent towards the object between $t \approx 200$ and $t \approx 400$. As before, the activation of sensor 5, at around $t \approx 409$, signals the end of phase 1 and stops motion. This time the decisive event appears to be the late activation of sensor 1, at $t \approx 556$. When this happens, the network outputs, already divergent, both change direction, eventually causing the agent to drive away at around $t = 700$.

In lesion tests it emerged that sensor 1 is important to both catch and avoid behaviors for controller A. In the pair of trials we have examined here, a decisive effect was observed for avoidance, but no effect was observed in the catch trial. This is because in the catch trial the inputs to the networks were already balanced by the time sensor 1 was activated. In this trial the removal of sensor 1 will cause an imbalance in the inputs to the networks at the point of activation for sensor 7, and therefore have a detrimental effect on performance.

5.3 Summary

We have seen that a certain behavioral variety is possible in the networks used here, but as with controllers A and A1, there are various features that all results seem to have in common. Due to the production of large signals in the network readouts and to the sigmoidal transfer function between readout and motors, agents tend to switch between being static and moving at maximum velocity. The effect of the sigmoidal function in this case could be construed as a rejection of redundant information. That said, although switching motors may appear crude, evolution has clearly exploited certain regularities in the visual field to produce controllers that switch at appropriate times across all the trials. In many cases, such as that of controller A, this behavior may not be very robust to disturbance, but we have seen from controller A1 that a more robust controller is possible, and that this appears to be because it reacts to stimuli over longer time scales.

6 Discussion

The controller in this experiment is analogous to a body with the simplest of nervous systems: weighted connections from sensory neurons to the body and weighted outputs from the body to two simple summing nodes. Hence the body performs the lion's share of the computation involved in producing adaptive behavior, which could be thought of as a reflexive response to different patterns of stimuli. Throughout most of this article it has been convenient to describe the two responses as catch and avoid, but, for example, we could equally conceive of this behavior as active perception in order to distinguish between friend and foe, resulting in stay or escape responses.

Due to the fading memory property, the networks used in this experiment have some capacity to store information about past inputs, but as yet there is no definitive evidence that this is exploited. While the state of the network certainly does play a central role in determining behavior, it appears that most controllers, like controller A, are tuned to react immediately to sensors being switched on and off. In the case of controller A1, its scanning motion implies integration of the streams from multiple sensors in a way that is unique in all our results. Closer scrutiny of input and output streams reveals that this controller does not respond to all sensory events, and that in some cases it actually seems to predict them and change direction just before its last sensor is switched off. The lack of a clear causal relationship between the various input streams and the network outputs for this controller suggests, at least, that the transient effects of inputs are longer-lived than in other controllers.

When the agent's visual field is given an information-theoretic interpretation, it appears that some controllers exploit paths of high information. In behavioral phase 1, where agents position themselves under the falling object, maximizing the information received by the sensors appears to be a sufficient and robust strategy, but for the decisive period of phase 2 it is not sufficient. This rules out the possibility of a controller that measures and acts to maximize the information at its inputs, and makes it clear that this is an adaptation at the evolutionary time scale. For example, the path of controller A in catch trials is such that a specific group of sensors are on during phase 1, and furthermore such that the gradient of increase of those stimuli is remarkably constant.

The information received from the visual field is first projected into the high-dimensional state space of the MSD networks; then there is a drastic two-stage reduction in information, first where the network state is reduced to a single readout output, and secondly when that output is fed

through the sigmoidal motor neuron, which thresholds the output and effectively rejects most of the information in that stream. An interesting line of enquiry that has not yet been addressed is to what extent this last stage is necessary for success.

It appears from the results of both perturbation and lesion tests that for this task there is no need to use a nonlinear MSD model. However, in biological materials, nonlinearity is the norm [34], so it would be interesting to adjust the nonlinear terms in Equation 7 so that nonlinear effects are magnified and see if successful controllers are still obtained. Our intuition is that this would in fact pose no problem, as the coupling between the MSD elements in the networks already leads to a high degree of nonlinearity.

7 Conclusion

According to Pfeifer and Bongard [29, p. 20], intelligence is "distributed throughout the organism" and not solely in the brain. The evidence for this is ever-growing, and we believe the work reported on here constitutes a significant addition to it. We have shown that compliant bodies with complex dynamics can integrate, store, and process information in meaningful and adaptive ways. If an abstracted model of a body and primitive nervous system can successfully perform adaptive reflexive behavior with the body as the main computational locus, then it seems reasonable to hypothesize that biological soft bodies could perform a similar function. Furthermore, if this behavior is of the nature of what has previously been described as minimal cognition, then the result challenges notions of brains as the ultimate and sole seat of intelligence and cognition.

Acknowledgments

C.J. was funded by a University of Sussex graduate teaching assistantship. A.P. and P.H. have received funding from the European Union's Seventh Framework Programme for research, technological development, and demonstration under grant agreement no. 308943.

References

1. Ashby, W. R. (1956). *An introduction to cybernetics*. London: Chapman and Hall.
2. Baratta, R., & Solomonow, M. (1990). The dynamic response model of nine different skeletal muscles. *IEEE Transactions on Biomedical Engineering*, *37*(3), 243–251.
3. Beer, R. D. (1996). Toward the evolution of dynamical neural networks for minimally cognitive behavior. In P. Maes, S. W. Wilson, & M. J. Mataric (Eds.), *From animals to animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior* (pp. 421–429). Cambridge, MA: MIT Press.
4. Beer, R. D. (2003). The dynamics of active categorical perception in an evolved model agent. *Adaptive Behavior*, *11*(4), 209–243.
5. Bongard, J. (2014). Why morphology matters. In P. A. Vargas, E. A. Di Paolo, I. Harvey, & P. Husbands (Eds.), *The horizons of evolutionary robotics* (pp. 125–152). Cambridge, MA: MIT Press.
6. Braitenberg, V. (1986). *Vehicles: Experiments in synthetic psychology*. Cambridge, MA: MIT Press.
7. Brooks, R. A. (1991). Intelligence without representation. *Artificial Intelligence*, *47*(1), 139–159.
8. Dale, K., & Husbands, P. (2010). The evolution of reaction–diffusion controllers for minimally cognitive agents. *Artificial Life*, *16*(1), 1–19.
9. Geyer, H., Seyfarth, A., & Blickhan, R. (2006). Compliant leg behaviour explains basic dynamics of walking and running. *Proceedings of the Royal Society B: Biological Sciences*, *273*(1603), 2861–2867.
10. Goldschlager, L., & Lister, A. (1982). *Computer science: A modern introduction*. Hertfordshire, UK: Prentice Hall International (UK).
11. Hauser, H., Ijspeert, A. J., Füchslin, R. M., Pfeifer, R., & Maass, W. (2011). Towards a theoretical foundation for morphological computation with compliant bodies. *Biological Cybernetics*, *105*(5–6), 355–370.
12. Hauser, H., Ijspeert, A. J., Füchslin, R. M., Pfeifer, R., & Maass, W. (2012). The role of feedback in morphological computation with compliant bodies. *Biological Cybernetics*, *106*(10), 595–613.

13. 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.

14. Husbands, P. (1998). Evolving robot behaviours with diffusing gas networks. In P. Husbands & J.-A. Meyer (Eds.), *Evolutionary robotics: Proceedings of the first European Workshop (EvoRobot '98)* (pp. 71–86). Berlin: Springer Verlag.

15. Iida, F., & Pfeifer, R. (2004). "Cheap" rapid locomotion of a quadruped robot: Self-stabilization of bounding gait. In F. Groen, N. Amato, A. Bonarini, E. Yoshida, & B. Kröse (Eds.), *Proceedings of the 8th International Conference on Intelligent Autonomous Systems (IAS-8)* (pp. 642–649). Amsterdam: IOS Press.

16. Johnson, C., Philippides, A., & Husbands, P. (2014). Active shape discrimination with physical reservoir computers. In H. Sayama, J. Rieffel, S. Risi, R. Doursat, & H. Lipson (Eds.), *Artificial life 14: Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems* (pp. 176–183). Cambridge, MA: MIT Press.

17. Klyubin, A. S., Polani, D., & Nehaniv, C. L. (2005). Empowerment: A universal agent-centric measure of control. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC 2005)*. Piscataway, NJ: IEEE.

18. Korn, G. A. (2005). Continuous-system simulation and analog computers: From op-amp design to aerospace applications. *IEEE Control Systems*, *25*(3), 44–51.

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

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

21. Lundberg, K. (2005). The history of analog computing: Introduction to the special section. *IEEE Control Systems*, *25*(3), 22–25.

22. Maass, W., Natschläger, T., & Markram, H. (2003). Computational models for generic cortical microcircuits. In J. Feng (Ed.), *Computational neuroscience: A comprehensive approach* (pp. 575–605). London: Chapman & Hall/CRC.

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

24. McGeer, T. (1990). Passive dynamic walking. *International Journal of Robotics Research*, *9*(2), 62–82.

25. Mirolli, M. (2012). Representations in dynamical embodied agents: Re-analyzing a minimally cognitive model agent. *Cognitive Science*, *36*(5), 870–895.

26. Nakajima, K., Hauser, H., Kang, R., Guglielmino, E., Caldwell, D. G., & 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*, 91.

27. Paul, C. (2006). Morphological computation: A basis for the analysis of morphology and control requirements. *Robotics and Autonomous Systems*, *54*(8), 619–630.

28. Pfeifer, R. (1996). Building "fungus eaters": Design principles of autonomous agents. In P. Maes, S. W. Wilson, & M. J. Mataric (Eds.), *From animals to animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior* (pp. 3–12). Cambridge, MA: MIT Press.

29. Pfeifer, R., & Bongard, J. (2006). *How the body shapes the way we think: A new view of intelligence*. Cambridge, MA: MIT Press.

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

31. Pfeifer, R., Iida, F., & Lungarella, M. (2014). Cognition from the bottom up: On biological inspiration, body morphology, and soft materials. *Trends in Cognitive Sciences*, *18*(8), 404–413.

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

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

34. Vogel, S. (2003). *Comparative biomechanics: Life's physical world.* Princeton, NJ: Princeton University Press.

35. Yamauchi, B., & Beer, R. (1994). Integrating reactive, sequential, and learning behavior using dynamical neural networks. In D. Cliff, P. Husbands, J.-A. Meyer, & S. W. Wilson (Eds.), *From animals to animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior* (pp. 382–391). Cambridge, MA: MIT Press.

36. Zhao, Q., Nakajima, K., Sumioka, H., Hauser, H., & Pfeifer, R. (2013). Spine dynamics as a computational resource in spine-driven quadruped locomotion. In *Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013).* Piscataway, NJ: IEEE.