University of Sussex

Moving softly - the role of morphological computation in the generation of intelligent behaviour

Chris Johnson

Submitted for the degree of Doctor of Philosophy

University of Sussex

January 2018

Declaration

I hereby declare that this thesis has not been and will not be submitted in whole or in part to another University for the award of any other degree.

Signature:

Chris Johnson

Acknowledgements

First, many thanks go to Andy Philippides and Phil Husbands for their excellent supervision. Thanks also to my examiners Chris Buckley and Fumiya Iida. Thanks to members of the overlapping groups of CCNR, EASy, and Alergic seminar speakers and attendees for many stimulating seminars and conversations. Finally, thanks so much to my parents for all of their support, especially over the last year.

UNIVERSITY OF SUSSEX

CHRIS JOHNSON, DOCTOR OF PHILOSOPHY

Moving softly - the role of morphological computation in the generation of intelligent behaviour

SUMMARY

The central theme of this thesis is 'morphological computation', in particular as it pertains to the generation of purposeful and intelligent behaviour.

The first part of the thesis covers experiments in simulations, aimed at exploring a previously unasked question: can genuinely computational soft-body reservoirs play a central role in generating behaviour which has previously been described as 'minimally cognitive'? We conclude in the affirmative, but also note that it remains unclear whether or how they can also exceed such abilities. The main part of this work has been presented at ALIFE 14 and ECAL 15, and published in the Artificial Life journal.

In part two, the focus moves to consideration of how morphological computation in soft bodies may collude with the action of nervous systems in the production of adaptive intelligent behaviour. The massive and hypnotic complexity of brains in present day species, and lasting traditions in the design and control of animals' mechanical analogies (robots), still lead many to perceive bodies in the neuromuscular species as being in service to the goals of their controllers (brains), but in fact the two are of a piece, and work together to the same ends. We hypothesise that this would have been more apparent than now at a time closer to the evolutionary introduction of neuromuscular systems, and introduce our new software framework for the evolution of virtual neuromuscular swimmers, which may be used to study artificial parallels to those distant origins. We begin with a demonstration of the technical innovations which we required in order to simulate soft-bodied swimmers in two-dimensional particle-based fluids, and then proceed to a description of our new concept for mapping arbitrary neural networks to arbitrary body morphologies.

Contents

List of Tables

List of Figures

. . . study hard what interests you the most
in the most undisciplined, irreverent
and original manner possible.

Feynman

Preface

Motivation

The germ that this thesis has grown from is the strange and initially unconvincing notion of bodies doing computation, introduced to the literature under the name of *morphological computation*. Morphological computation was originally introduced as a new perspective on *embodied cognition*. However, its role in behaviour has largely been investigated in the context of simplifying the control of what we might consider behavioural primitives, such as grasping and locomotion. As fruitful and illuminating as such investigations have been and continue to be, we believe that all behaviour, barring that which is pathological, is purposeful, from which it follows that all behavioural studies should take purpose into account. The central theme of this thesis is the following question: how far might soft bodies 'doing' morphological computation, or something related to it, play a part in the generation of purposeful and intelligent behaviour, rather than simply operating in its service?

Thesis overview

The thesis is split into two main parts. In the first part, we begin with a survey of the history of morphological computation, and the current conceptual state of affairs. This is necessary because, although the concept has caught on, especially in the soft robotics community, it is still lacking both a precise definition, and a solid foundation. After reviewing the positions of those relatively few researchers who have addressed the actual theory of morphological computation, we will set out our own. In short, this is that only some part of what is commonly referred to as morphological computation is easily defensible as being computational, but this is not to diminish the importance of the remainder, where morphology plays an important role in the spatiotemporal structuring and coordination of behaviour.

The first part of the thesis will continue with two chapters reporting on investigations conducted with one of the most convincingly computational morphological computation substrates to date, the mass-spring-damper networks introduced by Hauser et al. (2011). In Chapter 2 we apply these networks to a sensorimotor behavioural challenge, introduced by Beer (1996), which has been described as *minimally cognitive*. In Chapter 3 we proceed to a more demanding sensorimotor behaviour, introduced by Blynel and Floreano (2003), which explicitly requires the incorporation of memory of a binary variable for an extended period. In Chapter 4 we end the first part of the thesis with a description of some potential modifications to the networks which were partly responsible for the direction we took in the second part of the thesis.

In the second part of the thesis, the focus shifts from morphological computation specifically to more general considerations of the importance of morphology in the coordination of behaviour. We are developing a software framework which may be used to conduct a series of *in silico* experiments designed to study this in 'virtual creatures', introduced by Sims (1994). In Chapter 5 we describe the results of an experiment which began as a proof of concept for our simulation framework, but which upon closer examination led to some insights related to our thesis, and our introduction of a new subclass of morphological computation, which we have called *morphological*

coordination. In Chapter 6 we describe our proposed new approach to brain-body coevolution in soft-bodied virtual creatures. Finally, we close with a summary of the thesis, the implications of our results, and further directions for our own work and the study of morphology in behaviour in general.

Summary of contributions

The principal contribution of this thesis is to advance knowledge of the kinds of role morphological computation can play in the production of purposeful and intelligent behaviour. In order to do this, we centre our studies on a versatile and well understood variant of morphological computation, physical reservoir computing in mass-spring-damper (MSD) networks.

In Chapter 2 we show for the first time that MSD networks can be applied to controlling a mobile agent in an active vision task which is required to culminate in a decision between two courses of action. Previously, the applications of MSD networks, and physical reservoir computing in general, have not involved action selection and have only apprehended the environment indirectly via proprioceptive sensors. In Chapter 3 we take this further, applying our MSD network controllers to a far more challenging behavioural task for a mobile robot in a maze. The problem the robot was required to solve required navigation in the maze and also memory for a finite but extended period, where the robot was expected to return to the location of a target which was discovered earlier. As well as giving our controllers a more demanding task, we also tested their robustness by requiring them to cope with both motor and sensor noise, and some random variation in the dimensions of the maze. MSD networks have fading memory, which is advantageous in applications such as filtering, but which in other contexts may be a disadvantage. One solution to overcoming the fading memory limitation is with a feedback loop. Also in Chapter 3, we introduce a novel solution to bistable and persistent memory elements in MSD networks, a potentially useful addition to our box of morphological computation tricks.

Having shown that MSD networks can be used in the generation of purposeful and intelligent behaviour of mobile agents, we are now interested in seeing how far this is the case when the bodies of said agents *are* the MSD networks. The context we have chosen for investigating this is soft-bodied swimming virtual creatures. We consider fluidic environments particularly rich for studying the role of morphological computation in behaviour, as both the bodies of our swimmers and the environment they exist in are soft. This is not the case when simple drag based fluid models are used, and so to increase the range of fluidic dynamics and therefore body-environment interactions, we have introduced particle-based fluids to the virtual creatures domain. In Chapter 5 we describe our novel methods for the use of Google's *liquidfun* (Google, 2013) particle-based physics engine to simulate what we call *pseudo-soft*-bodied swimmers in particle-based fluids. Also in Chapter 5, we introduce our new subclass of morphological computation, *morphological coordination*, which we argue can clarify certain aspects of morphological computation theory which remain poorly defined.

Finally, in Chapter 6 we briefly look at where our work is headed, with a focus on novel methods which we are currently developing for mapping neuromuscular networks to bodies with arbitrary morphologies which we believe may lead to advances in virtual creatures.

Publications and conferences

- The work in Chapter 2 was presented at ALIFE 14 (The 14th International Conference on the Synthesis and Simulation of Living Systems) (Johnson et al., 2014), and subsequently published in Artificial Life (Johnson et al., 2016).

- Preliminary results from Chapter 3 were presented at ECAL 15 (The 13th European Conference on Artificial Life) (Johnson et al., 2015).

- A version of Chapter 5 has been accepted for publication in Soft Robotics, under the title *Simulating soft-bodied swimmers with particle-based physics*.

Part 1 - Minimal cognition in mass-spring-damper networks

Chapter 1

Literature review

In this chapter, we briefly cover the history of the idea, *morphological computation*, and the aspects of the theory that has grown around it which are most relevant to our own thesis. We introduce the concept of *control-centrism*, as the counterposition to our thesis, and define a new subclass of morphological computation, *morphological coordination*, which we argue is a more appropriate way to explain some of what is currently referred to as computational.

1.1 Introduction

The first part of this thesis is directly focussed on the subject of morphological computation. The following chapters recount a series of experiments directed at expanding our knowledge of the behaviour-generation capabilities of reservoir computers with a morphological computation substrate, the mass-spring damper network (MSD network). We will describe the networks themselves in Chapter 2, but before proceeding will give some space to the background and meaning of morphological computation itself. This is necessary for two main reasons: firstly, the idea of morphological computation was introduced without a precise definition, and few authors have since attempted to make one. Secondly, the use of the word 'computation' in this context has proven controversial. A number of researchers, while generally interested in the phenomena referred to as morphological computation, are dismissive of the designation 'computational'.

In this chapter, we will first summarise key results and theoretical additions that have been either described as or connected to morphological computation, and what they have shown. We will then give some space to a critique of the concept itself. While there is much related research, from both before and after the introduction of the morphological computation concept, we will focus only on that which makes explicit reference to it, in order to avoid the scope of this chapter from becoming too broad.

1.2 Control-centrism

One of the central themes of this thesis is the idea that a lot of the work conducted under the banner of *morphological computation* can be construed as a challenge to what we refer to as control-centrism. By control-centrism we mean a principle, of modelling or design, whereby a hard line is drawn between controller and controlled, and the same boundary is assumed to define the home of intelligence. Mind-body dualism could be considered a form of control-centrism, but the two are not synonymous, as the idea of control-centrism can be applied to robots and animals regardless of the presence or absence of mind.

Although there are a large and ever-growing number of counter-examples, some of which we will introduce in the coming sections, it has long been the norm for robot designers to build the body of a robot first, which will have the potential for certain capabilities, and then embed the

robot's purpose and the instructions required for its achievement in the robot's controller. The controller manipulates the body of the robot and its environment to achieve the purpose, and, crucially, both initiates behaviour and evaluates its effects. Everything that is intelligent about the robot is in the controller, and in a sense the body is there to serve it. Through the analogy between a robot controller and a nervous system, natural intelligence has come to be viewed in a similar light.

The centrality and over-arching importance of control is too often assumed, rather than justified, as being the state of affairs in intelligent behaviour, even when the dynamics of the body and the world are emphasised as influencing behaviour. We argue that this assumption should not be made as there are other possibilities. For example, as in the quadruped robots of Iida (2005), which we will look at in Section 1.3.3, an open loop controller may do little more than inject energy into the system. In this case, not much can be learned about what the robot does, or how, by studying the control signals. This is only possible by studying the dynamics of the body. Therefore, in at least some cases, it seems just as appropriate to consider the controller as serving the body as vice versa.

The contrast between morphological computation and control-centrism in the production of behaviour is related to that between the two different approaches to motor control discussed by Schaal et al. (2007), dynamics systems and optimal control. Almost all of the work described in this chapter, and all that is presented in this thesis essentially falls into the dynamics systems camp. Traditional optimal control is more closely aligned with the control-centric point of view, as it assigns policies which map from states to actions. In the morphological computation approach to behaviour, the tendency is more to seek for appropriate attractors and limit cycles inherent in the dynamics of body-environment interactions, and to use them to minimise control requirements.

1.3 Summary

In this section, we summarise some key results and theories related to morphological computation, selected according to their relevance to this thesis.

1.3.1 Paul

One of the first explicit references to 'morphological computation' in print is in the PhD thesis of Paul (2004). Therein, Paul sets out to shed new light on an idea earlier introduced by Pfeifer and Scheier (1999), the *tradeoff* between morphology and computation, by showing how a '...morphology can subsume a computational role'. Paul illustrated the concept with thought experiments where parts of the bodies of robots can form an integral part of the computation of logical functions, and referred to them as '...an important proof of concept, that morphology can perform computation which can reduce the computational requirements of the controller'. As Paul admitted, the concept of morphological computation was not well developed at this point, and subsequent interest in this logical form of morphological computation has been low. However, as we shall see, the idea of morphology taking over some computational functions which may otherwise belong to controllers became axiomatic.

1.3.2 Pfeifer et al.

In the following years, the idea of morphological computation was elaborated upon and illustrated using numerous examples by Rolf Pfeifer and others in a series of articles and a book (Pfeifer and Iida, 2005; Pfeifer et al., 2006; Pfeifer and Bongard, 2006). In these works, morphological computation was presented as a more general concept related to embodied intelligent behaviour, which '...can be exploited on the one hand for designing intelligent, adaptive robotic systems, and on the other hand for understanding natural systems.' (Pfeifer et al., 2006).

Broadly speaking, two types of morphological computation are discussed. The first is a kind of preprocessing of sensory stimuli, the effect of sensor morphology on the informational structure of sensory streams. The second is related to the influence of morphology on action. For example, according to Pfeifer and Bongard (2006), 'One especially interesting form of morphological computation is provided by the intrinsic dynamics of the physical system, i.e., its attractor dynamics. Exploiting this dynamics can lead to the achievement of tasks entirely "for free," with no control. The passive dynamic walker is a prime example: it demonstrates how a system self-stabilizes while walking as long as its morphology and environment are appropriate, or to use dynamical systems jargon, as long as it is in the basin of attraction of the attractor for walking.'. This is an important example for two main reasons: first, the passive dynamic walker is an extreme case, as, although it can only do one thing, and that in only a narrow ecological niche, it does what it does in the complete absence of control. Secondly, this may be the first explicit mention, in print, of the connection between morphological computation and dynamical systems concepts such as basins of attraction.

One other thing which stands out in these early works is a certain amount of equivocation over how far morphological computation is genuinely computational. For example, Pfeifer et al. (2006) wrote: 'It should also be noted that these motor actions are physical processes, not computational ones, but they are computationally relevant, or put differently, relevant for neural processing, which is why we use the term "morphological computation".', but later added 'Another problem is that the notion of computation in the context of morphology or dynamics may in fact require fundamental reconceptualization, which is in itself a challenging research topic.'. In the first statement, it seems to be made clear that the processes of interest are not computational, although they may engage with processes which are. The second quote is more interesting, as the computational aspect is not questioned here, although there is half an admission (in the words 'may in fact') that it has not been well justified.

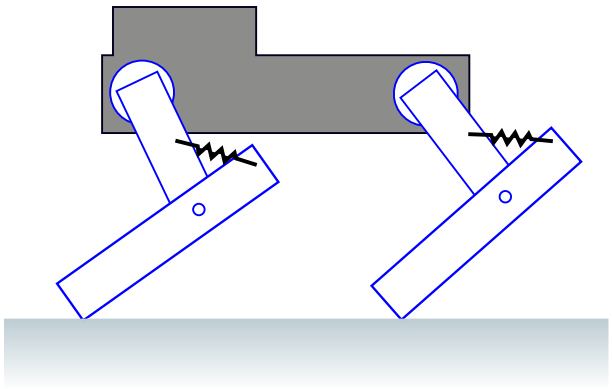


Figure 1.1 A quadruped robot, seen from the side (adapted from (Iida, 2005)). The stability conferred by the addition of springs to make the leg joints compliant makes it possible to control the robot with a simple feedforward controller.

1.3.3 Softness

Although, as evidenced by the thought experiments of Paul (2004), morphological computation can be achieved with simple wheeled robots, it has most often been demonstrated with robots which show some degree of compliance, ranging from robots with compliant joints to robots with soft and fully compliant bodies. As robot bodies become softer, they become better analogies to the bodies of animals, and the implication of this is that if soft robots have properties facilitating morphological computation, then animal bodies will too.

For example, the quadrupeds from Iida (Iida, 2005; Pfeifer and Iida, 2005; Pfeifer et al., 2006)(Figure 1.1) were capable of bounding gait locomotion while controlled by two simple os-

cillators. They were able to do this largely due to the stability and energy storage and release conferred through the use of springs to make leg joints compliant. This simple biomechanically-inspired mechanism reduces the control requirement for locomotion as it constrains the body's dynamics.

Tensegrity robots have been used to make a similar point. From the early examples of Paul (2006) through to more recent ones from Khazanov et al. (2013)(Figure 1.2) it has been made clear that for the cyclical actions which lead to locomotion, the dimensionality of the controller signals may be far lower than the dimensionality of the body. While their forms may appear abstract, tensegrity robots are of particular interest because of their connection to *biotensegrity* (Scarr, 2014), which has been put forward as fundamental to the structure of life and could eventually lead to a radical overhaul of biomechanics.

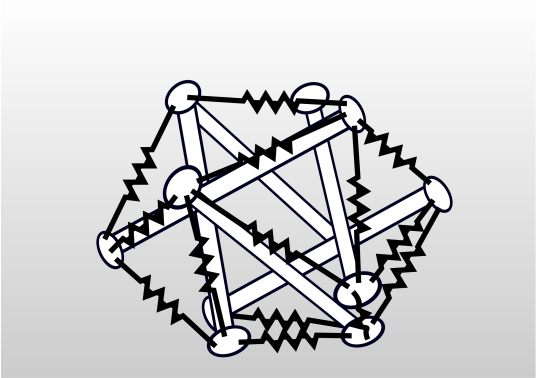


Figure 1.2 A tensegrity robot (adapted from (Khazanov et al., 2013)). The tensile elements on this tensegrity are springs, the resonant properties of which are exploited for fast locomotion generated by vibration.

1.3.4 Morphological communication

A significant result came from Rieffel et al. (2010), based on a simulated tensegrity robot. Paul (2006) had earlier connected morphological computation to tensegrity robots, and shown that these high-dimensional structures with complex dynamics could be effectively controlled in locomotion with only a low-dimensional control signal vector, but Rieffel et al. demonstrated something new, the body of a robot acting as a communication channel. Watanabe and Ishiguro (2007) had already shown, using a simulated serpentine robot, how increasing the length of mechanical couplings along the body could make learning gaits easier, and they had connected their observations to morphological computation. Rieffel et al. described a similar phenomenon, but were able to clearly demonstrate the evolution of communication between independent spiking neural networks via dynamical couplings in the body. The appellation of morphological communication for this phenomenon does not seem to have caught on, but this is an important result as signalling is fundamental to the organisation of intelligent behaviour, and the morphology of the body does more here than just make it easier to control by conferring stability or granting affordances for environmental interactions.

1.3.5 Mass-spring-damper networks

In 2011, Hauser et al. made what may be the most significant addition to the theory of morphological computation since the early publications. Although the title of their article, 'Towards a theoretical foundation for morphological computation with compliant bodies', suggests a more far-reaching importance to this paper than may be justified, it does seem to have re-energised theoretical approaches to the subject. What Hauser et al. presented in this paper is not a general theory which might explain or define all morphological computation, or even all morphological

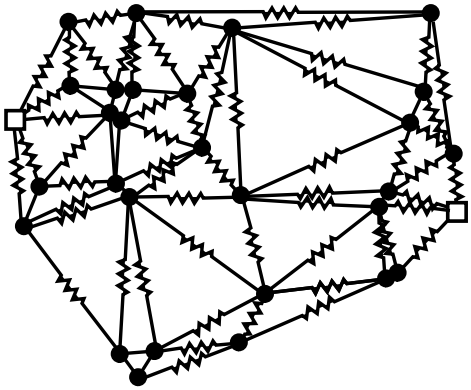


Figure 1.3 An example mass-spring-damper network, from Chapter 4 of this thesis. Dampers are not shown, for clarity, but the motion of every spring is in fact damped.

computation in compliant bodies, but rather a strong proof for a certain class of morphological computation, which is of clear practical use to robot designers, and has strong implications for biology.

Hauser et al. showed that a simulated network of mass-spring-dampers (Figure 1.3) could be used as a physical reservoir to perform complex mappings from input to output streams. Whereas in echo state networks (Lukoševičius and Jaeger, 2009), where fading memory has to be enforced, it is guaranteed in mass-spring-damper networks, assuming all elements have non-zero damping. Hauser et al. demonstrated that, despite this property, their networks could emulate complex filters which require integration over time, such as Volterra series operators (we will return to the Volterra series in Chapter 4). As long as a network has a rich enough set of dynamics, the system can be trained relatively easily by setting the weights of a linear readout. Importantly, an arbitrary number of readouts can be attached without influencing the network dynamics, or each other, meaning that a single system can be trained to apply multiple transformations of a single input stream in parallel.

In a following paper, the same authors (Hauser et al., 2012) also showed that, with the addition of a feedback loop, the same networks could emulate dynamical systems attractors, and generate cyclical signals similar to central pattern generation for robot control. As we shall see in the following section, this technique has also been put to good use with real robots.

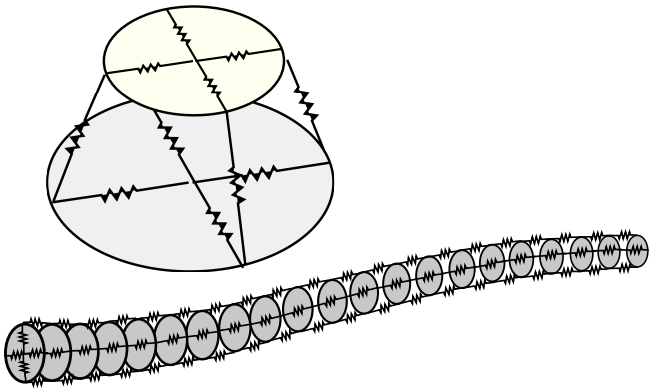


Figure 1.4 A biologically plausible model of an octopus arm (adapted from (Nakajima et al., 2013)). The arm is constructed from hydrostatic muscular cells, constructed using mass-spring dampers. A single cell is shown above an arm made from 20 cells.

1.3.6 Physical reservoir computing with robots

Nakajima et al. (2013) showed that the same filters and cycles implemented by Hauser et al. with two-dimensional mass-spring-damper networks can also be achieved with a three-dimensional bio-

mechanical model of an octopus arm, also constructed out of mass-spring-dampers (Figure 1.4). Not only did they extend the networks into the third dimension, but they used biologically plausible parameters in the model. Nakajima et al. (2015) later showed that a robotic octopus arm, with a reservoir of bend sensors embedded in silicone, could be used in a similar way.

Zhao et al. (2013) embedded a reservoir of force sensors in the flexible spine of a quadrupedal robot, and trained it to control the robot in locomotion. This was a particularly nice result, as the robot was actuated solely by the same spine, meaning there is no meaningful distinction between body and controller in this robot. Caluwaerts et al. (2013) applied a similar reservoir computing approach to control simulated tensegrity robots for locomotion and end-effector motion. More recently, Eder et al. (2017) have applied the same techniques to the controlling endpoint trajectories of a highly complex robot arm. A task like this typically involves complex kinematics, but as the reservoir computing approach to training the system was followed, no model of the arm was required.

It is worth noting that in all of these examples, the reservoirs are purely proprioceptive, accessing only the body's state directly. The environment can only be detected indirectly in this case, inasmuch as the interactions between body and environment can affect the state.

1.3.7 Theoretical additions

While the reservoir computing approach to morphological computation is well defined, the more general concept is generally defined quite loosely, in particular as it pertains to computation. The most rigorous attempt at a formal definition of morphological computation yet may be that of Füchslin et al. (2013). Of the points they make, one of the most relevant to this thesis is related to control. They state 'In our terminology, "morphological control" means "control achieved via morphological computation."'. They note that in control theory, the central notion is that of a *reference*, but that no explicit reference is required in morphological control. We will return to the subject of morphological control in Section 1.5, but for now say that we believe that the real significance of this paper is that, in its attempt to be precise about morphological computation, it exposes some of the issues with the concept. For example, it seems to exclude the example, still being given by Pfeifer et al. (2014), of morphological computation as the preprocessing of stimulus information by sensor morphology. It also draws attention to the fact that the inclusion of the passive dynamic walker in examples of morphological computation is problematic, although that may not be the view of the authors.

Müller and Hoffmann (2017) have considered some of these issues in detail. They begin by asking: does appropriate use of body morphology reduce the need for computation, or does it distribute it between controller and body? They dismiss the idea of computation in the passive dynamic walker, and robots like Iida's quadrupeds. Like Füchslin et al., they prefer the term 'morphological control' here, but they differ in arguing that it is 'morphology facilitating actuation' rather than 'control achieved via morphological computation.'. Similarly, they discuss 'morphology facilitating perception', and, ultimately, only consider the physical reservoir computing approach to morphological computation to be genuinely computational.

1.4 Outsourcing and exploitation

The next theoretical issue we will consider is one of perspective. There are far more allusions to morphological computation in biology (from engineers and computer scientists) than there are concrete examples, but there are also some oft-touted aspects of the theory which we believe reveal the perspective of a robot designer, and are less useful in a biological context.

The first is the notion of computation being 'outsourced' (Nakajima et al., 2013) or 'offloaded' (Pfeifer and Bongard, 2006) to the body. Müller and Hoffmann (2017) have already questioned whether computation per se is being delegated to the body at all, but for now we will reserve

judgement on this point, which we will come back to in Section 1.5 The idea of outsourcing may make sense from the point of view of a designer, but also ultimately seems control-centric. For example, adding compliance to a robot's joints may constrain its dynamics and make it more stable, thereby reducing the control requirement in the traditional computational sense. In this case, it may make some sense to talk about offloading from the controller, because we know the progression that has taken place in terms of design. On the other hand, to follow a similar logic, if a purely mechanical automaton like the passive dynamic walker (McGeer, 1990) is turned into a robot through the addition of a controller in order to improve its walking capabilities (as has indeed been done, for example by Tedrake et al. (2004)), then should we take it that computation has been outsourced from the body to the controller? If we discuss outsourcing to the body in both cases, then we must have assumed in advance the central importance of the controller.

A related concept is the idea of 'exploiting' the dynamics of the body in order to reduce control requirements (Pfeifer et al., 2006). Again, this only really makes sense from a point of view which is a designer's, or control-centric, or both. Who or what is supposed to be doing the exploiting in nature?

1.5 The three Cs: computation, control, and coordination

1.5.1 Computation

Müller and Hoffmann (2017) argue that most of what is called morphological computation is not genuinely computational, and only physical reservoir computing really deserves the name.

On the subject of morphological computation, Pfeifer et al. (2014) have recently stated 'An important implication from the examples above is the fact that computation or information processing in the Turing or Shannon sense needs to be extended . . . '.

The earlier concept of the tradeoff between morphology and computation still makes sense, but to an extent, extending it to the idea of outsourcing computation was done prematurely. Before morphological computation was properly defined, it was being pointed to in the bodies of many robots and animals.

As pointed out by Müller and Hoffmann, one apparent solution to this problem is to take recourse to ideas such as 'pancomputationalism', the idea that everything computes. On such a broad definition of computing, we are inclined to agree with Harvey (1997), who wrote: 'There is a regrettably [sic] tendency to use the word "computation" to refer to the workings of any complex system, but this results in the word losing any useful sense, and such silly claims as a thunderstorm, or indeed the universe, is *computing* its next state.'. Even assuming that an extended theory of computation can be devised, that doesn't necessarily mean it will be the most useful way to explain all of what is now referred to as morphological computation.

1.5.2 Control

If we consider the notion of morphological control given by Füchslin et al. (2013), it inherits all the issues around the idea of morphological computation.

If we consider that of Müller and Hoffmann (2017), it seems more intuitive and easier to justify. However, we don't agree that the passive dynamic walker should be included in this category, as it has no controller. Stability of a system, whether static or dynamic, does not necessarily imply that the system is being controlled, i.e. stability may be due to other conditions.

1.5.3 Coordination

To the list of morphological computation, morphological control, and morphological communication, we propose the addition of morphological coordination. We will explain this concept further in Chapter 5, with examples including our own swimming virtual creature, but for now will return

to the passive dynamic walker. This automaton has often been pointed to as a prime example of the power of morphological computation, as it walks without the need for any controller. However, we believe that it makes no sense to discuss control in the absence of a controller, and computation is also difficult to justify in this case. Morphology can constrain and *coordinate* the motions of a body in such a way as to facilitate behaviour, and this is what we term morphological coordination.

1.6 Conclusion

While there are clear issues, which we have only touched upon here, with how exactly morphological computation should be explained and even named, there is no doubt that the label is being applied to related areas which are of much interest. Broadly speaking, as long as we focus only on morphological computation in behaviour, then we are in the domain of embodied cognition, with an emphasis on the role of the body's morphology. We have briefly introduced our new subclass of morphological computation, *morphological coordination*, which is related specifically to the effect of morphology on behaviour. We explain this idea further in Chapter 5.

Physical reservoir computing, including with MSD networks, has been applied to filtering, central pattern generation, and controlling behavioural primitives such as locomotion and end-point trajectories. In all the robotic examples, the reservoir is made up of proprioceptive state sensors. In the following two chapters, we apply MSD networks to controlling purposeful and intelligent behaviour for mobile agents, incorporating exteroceptive sensing for active vision, decision making, navigation, and memory for a finite but extended period.

Chapter 2

Decision making with mass-spring-damper networks

This work described in this chapter was presented at ALIFE 14 (The 14th International Conference on the Synthesis and Simulation of Living Systems) (Johnson et al., 2014), and subsequently published in Artificial Life (Johnson et al., 2016). We introduced a novel physical reservoir computing controller, based on mass-spring-damper (MSD) networks (Hauser et al., 2011) for a mobile agent engaged in reflexive behaviour which has previously been referred to as *minimally cognitive*. The experiment which we used to test our controllers, introduced by Beer (1996), requires active sensing of objects in the environment and decision making, challenges which MSD networks had not previously been presented with. We noted in Chapter 1 that in all the robotic examples we gave of physical reservoir computing the reservoirs are entirely made up of proprioceptive sensors, and so these robots can only detect the environment indirectly via the effects of the body's interactions with it. Here, we connect simple vision sensors directly to the inputs of our reservoirs, so that our controllers are directly sensitive to the environment and the interaction between motion and vision. We show that our controller design and the dynamics of MSD networks are both up to the challenge, and amenable to 'programming' through the use of an evolutionary algorithm.

2.1 Introduction

The importance of embodiment in both generating and understanding adaptive behaviour has been increasingly recognised over recent years (Braitenberg, 1986; Brooks, 1991; Pfeifer, 1996; Pfeifer and Bongard, 2006). 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 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 et al., 2014). Pfeifer and Iida (2005) 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 under-explored, 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 chapter 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 (RC) (Maass et al., 2003; Lukoševičius and Jaeger, 2009), an approach they demonstrated with simulations of recurrent networks of mass-spring-damper (MSD) elements.

The passive dynamic walker (McGeer, 1990) 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 as 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 minimisation 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 (Iida and Pfeifer, 2004; Pfeifer and Iida, 2005; Pfeifer et al., 2014).

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 (Paul, 2006; Valero-Cuevas et al., 2007), 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 (Goldschlager and Lister, 1982). However, running parallel to what has, perhaps irreversibly, become the mainstream of computing, is a tradition of analogue computing (Lundberg, 2005; Korn, 2005). For decades, analogue computation was the only feasible option for the simulation and control of complex systems in applications such as aeronautics and spaceflight (Korn, 2005). 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 (2005) is of the same kind and so legitimised 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) (Korn, 2005; Ashby, 1956). Therefore there is a direct connection between an MSD network and the very origin of analogue computing. In the remainder of this chapter, where we use the word 'computation', we will be referring to analogue computation.

Hauser et al. (2011) 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 non-linear 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 realisable and because of their similarity to biomechanical muscle models (Hill, 1938; Geyer et al., 2006; 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 be trained to perform pattern generation without the need for external stimulation. Nakajima et al. (2013) 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. (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 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 which do not reach its proprioceptive force sensors.

We will show that morphological computation can go further. If continuous-time neural networks (CTRNNs) (Yamauchi and Beer, 1994), GasNets (Husbands, 1998) and other dynamical neural models can generate adaptive behaviour, 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 behaviour, rather than merely being in the service of a central controller.

In order to test whether MSD networks can generate adaptive behaviour we selected an experiment introduced by Beer (1996), in which an agent controlled by a small continuous-time recurrent neural network (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 has been made much of (Beer, 2003; Dale and Husbands, 2010; Mirolli, 2012). 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 behaviour exhibited by our agents as being of the reflexive variety, with the choice made by the agent being to either initiate escape behaviour or not in response to different patterns of stimuli.

In the rest of the chapter we show that we can evolve MSD networks that are capable of generating adaptive behaviour, 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 behaviours 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 of how the networks actually perform their function with perturbation and lesion tests, and with case studies of two controllers which generate behaviour which is representative of the two far ends of the observed behavioural spectrum.

2.2 Methods

2.2.1 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 (see Figure 2.1 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 behaviours 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 behaviour on the right-hand side of the arena 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 horizontal velocity is the sum of the two opposing motor outputs. The transfer function for the motor pair is given by:

$$5[\sigma(N_r + \theta) - \sigma(N_l + \theta)] \tag{2.1}$$

$$\sigma(x) = 1/(1 + e^{-x}) \tag{2.2}$$

where N_l and N_r are the outputs from the left and right MSD network readouts, respectively, and θ is a constant which 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 and the use of a multiplier of 5 on the result of the sum specifies a horizontal velocity in the range of $[-5, 5]$.

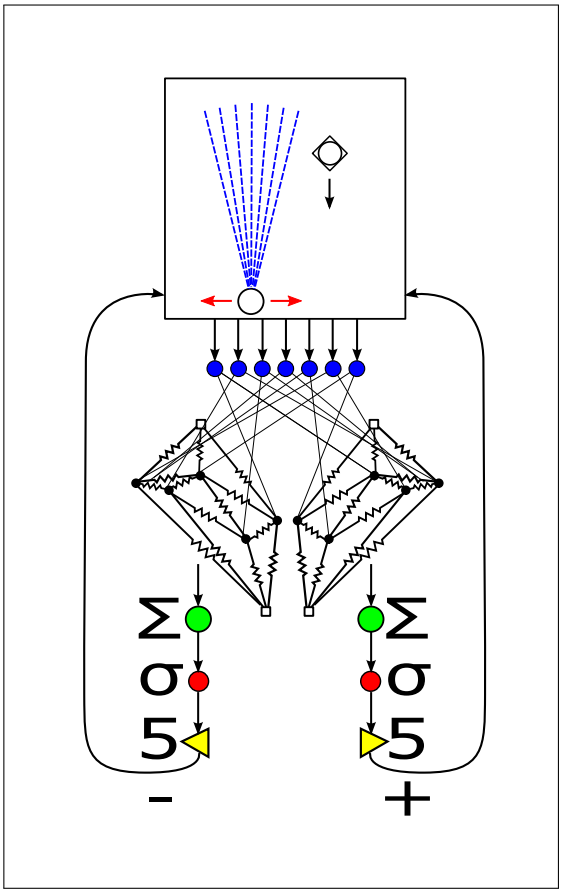


Figure 2.1 The agent-environment coupled dynamical system. The agent in the environment, including falling objects, is shown in the panel, and the connected controller is shown below. The agent moves left and right, to catch objects which are circle shaped and avoid those which are diamond shaped. Falling 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.

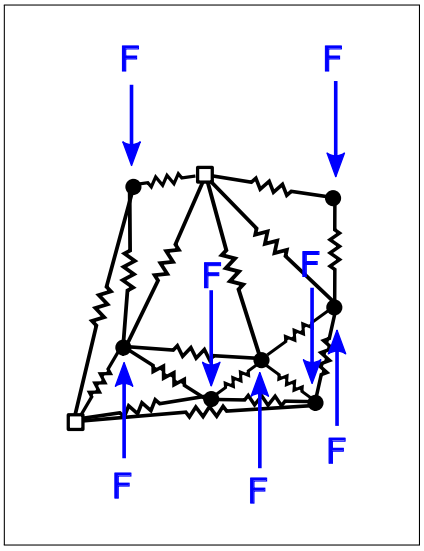


Figure 2.2 An MSD network. The nodes of the network are point masses, connected by parallel springs and dampers. The two most distant nodes, marked with squares, are fixed, while other nodes are free to move. Inputs to the network take the form of forces applied to all or a subset of the free nodes, parallel to the vertical axis. (Dampers and some springs and nodes are ommitted for clarity.)

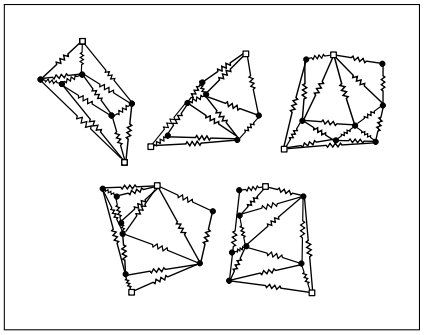


Figure 2.3 Topologies of the first successful evolved controllers. From left to right, the top row shows controllers A, B and C, and the bottom row shows controllers D and E. (Dampers and some springs and nodes are ommitted for clarity.)

The agent's sensors are 7 rays uniformly spaced across an angle of $\pi/6$ radians and centred 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:

$$\tau_i \dot{s}_i = -s_i + I_i(x, y) \qquad i = 1, \ldots, 7, \tag{2.3}$$

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.

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.

2.2.2 Mass-spring-damper 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.

The spring networks used here are based upon those in Hauser et al. (2011). The springs are connected to each other in a 2-dimensional plane (Figure 2.2). 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:

$$\dot{x_2} = \dot{x_1} \tag{2.4}$$

$$p(x_1) = k_3 x_1^3 + k_1 x_1 \tag{2.5}$$

$$q(x_2) = d_3 x_2^3 + d_1 x_2 \tag{2.6}$$

$$\dot{x_2} = -p(x_1) - q(x_2) + u, \tag{2.7}$$

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 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:

$$\dot{x_2} = \dot{x_1} \tag{2.8}$$

$$\dot{x_2} = -\frac{k}{m}x_1 - \frac{d}{m}x_2 + \frac{1}{m}u, \tag{2.9}$$

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 (2.7), 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 Equation (2.9) is simplified to a form similar to Equation (2.7).

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 (2.7) or Equation (2.9). 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 2.2. 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 sensor neurons (Figure 2.1). 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. (2011) here. We use the spring extension in the output sum where they used the overall length, and added extension velocities as a computationally cheap way to potentially increase the information encoded in the network output. As the network is always started from its resting state, i.e. with all spring extensions and extension velocities at zero, this means that the network output is also initially zero, although sensory inputs soon change this state of affairs. The outputs of the two networks are fed into the motor function Equation (2.2) 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.

2.2.3 The evolutionary search algorithm

Some network parameters are generated randomly and others are set through 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. 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:

$$W_{i,j} = \frac{f(p_i) - f(p_j)}{|p_i - p_j|} \tag{2.10}$$

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 i-th individual, and $f(p_i)$ is the fitness of the phenotype which 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:

$$p_i(t+1) = p_b(t) + \rho\lambda[p_b(t) - p_i(t)] \tag{2.11}$$

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 (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 $\rho = 0.3(1 - f_{max})$, 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 (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 Equation (2.2). 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 2.3.

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 normalised between 0 and 1. Hence 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.

2.3 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, due to the fact that 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 non-linear 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 (2.2) (Table 2.1, column 1). The ranges of all evolved parameters are given in Table 2.2. 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 which led to a number of successful controllers are given in Table 2.3.

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 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). Briefly, non-linear spring coefficients are drawn from a uniform distribution in the interval $[100, 200]$ and linear

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 2.1 The first winning combinations discovered. 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.

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 2.2 Limits placed on evolved parameters for configurations A through E.

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 favours 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 of ~98% of the perfect score was determined to be sufficient to ensure the correct behaviour for all trials, and 5 out of 20 runs resulted in viable controllers. Table 2.1 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 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 configuration of controller A, selected because it gives the highest dimensional search space without using feedback. While in many search algorithms increasing dimensionality is an issue, in evolutionary algorithms it can be advantageous in that it may introduce more pathways to success (Chen and Conrad, 1994; Barnett, 1998; Bongard, 2014). Runs continued until the search could be seen to either have succeeded or effectively halted at a local optimum, up to a maximum of 1000 generations. In this case only 4 runs met the success criterion, but the average fitness was high, as can be seen in Table 2.3.

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 2.3). The first set of 20 was with a configuration which will be referred to as 'J', which had the input weight range doubled to $[-4, 4]$. This change was made based on an observation that controllers which just failed to succeed always did so because they did not respond correctly to the most distant falling objects, which are initially only detected by the outermost sensors, and that only briefly 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 such 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 box plot in Figure 2.4 graphs 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 alone makes J and W look strongest, but when we look at other statistics such as number of runs which succeed according to the 98% criterion, and how long it takes to obtain those successes (Table 2.3), a slightly different picture is painted. Configuration S, perhaps because it has a 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 too much can be drawn from the mean time to success for all configurations, but they seem comparable. It may be that a further improvement would be made by modifying configuration S to use the same input weight pattern as in configuration W, which would 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.

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

Table 2.3 Statistics of evolutionary runs for configuration A and derived configurations. The statistics are calculated from the results of 20 runs for each configuration.

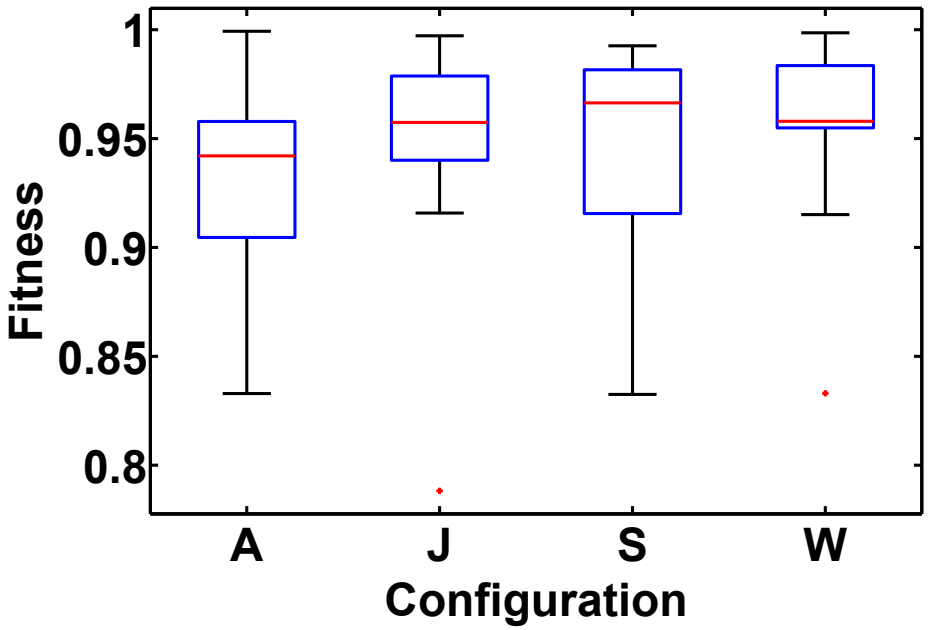


Figure 2.4 Fitness distributions for controllers with configurations A, J, S, and W, over 20 evolutionary runs for each configuration.

In summary, a number of different configurations of MSD networks and network layouts (Figure 2.3) have proven effective. Interestingly, there is also considerable variety in the behaviour 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 2.5 we show an example of each. The individual figures plot the horizontal distance between agent and object over time. When the agent's behaviour 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 behaviours arise will help us to understand how compliant networks can generate adaptive behaviour. The following section will thus analyse the behaviour of MSD networks in this task in general, and finally we will analyse two specific controllers in detail in Section 2.5.

2.4 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 maximise information from the sensors. We examine the capacity for memory of the networks 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

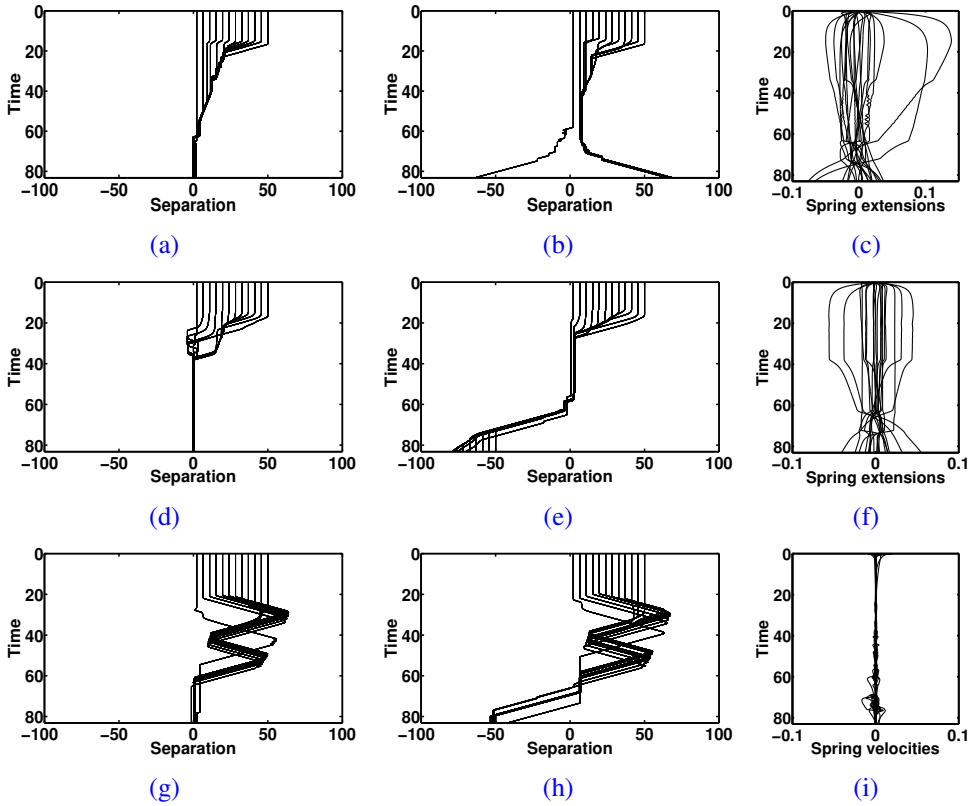


Figure 2.5 (a), (b), (d), (e), (g), (h) Agent trajectories throughout all evaluation trials. Trajectories from catch trials are shown on the left and those from diamond trials on the right. Trajectories from three controllers are shown: from top to bottom, controllers A, A5 and A1. In all cases the behaviour may be split into two phases, a 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 A1 in trial 19 (avoid). Note that although the agent is in motion for most of the trial, as seen in (h), the spring velocities are very close to zero.

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 behaviour 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 successful MSD controller which we have yet seen which scans objects in a similar fashion to Beer's CTRNN (Beer, 1996, 2003).

Before continuing to more detailed scrutiny of individual controllers, there are certain observations which 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. Due to the fact that sensory input is normalised 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. Although the MSD networks have non-zero output throughout this period, 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 2.15c). The initial biased configuration takes

the form of energy stored in the networks, and so as the inputs rise towards 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 2.5c 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 behaviour of the agent over the remainder of the trial can be split into two phases (Figure 2.5). In the first phase the agent positions itself under the object, not necessarily centrally or statically, as we shall see in Section 2.5, 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 2.1). Circular objects are 30 units wide, but although diamonds are squares of 30 units wide, they are oriented such 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, a difference which 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 which 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 which is ripe for exploitation by the evolutionary process in producing successful behaviour.

2.4.1 Information analysis

In this section we consider an information-theoretic explanation for the behaviour 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 such 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 maximising sensory information, for at least some of the time. In the foregoing we explore how far this is the case, and at which timescale this adaptation takes place.

For each trial, we can predict exactly what the agent will 'see' at any point in time and from any horizontal position that it can reach. This is because the agent's sensors are unaffected by noise, and the object falls vertically at a constant rate at a predetermined horizontal coordinate. 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 analysed for informational content. The measure we use is self-information (Jones, 1979), often referred to as surprise, defined for a given event as the negative logarithm of the probability of the event's occurrence. The use of the logarithm ensures that self-information is always non-negative, is additive, and that information is high for low probability events, and vice versa. To measure the self-information of the patterns of stimuli in the lookup table, we begin by quantising the stimuli levels in the tables to 200 distinct levels. For each lookup table entry (representing the sensory input for an agent at position x and time t) we treat the sensory vector of 7 stimuli as a single event, and then calculate the probability of occurrence, over the entire table, for each event. From this probability mass function we then compute the self-information, I, of each table entry, using the following equation:

$$I((x,t)_k) = -log_2(p_k) \tag{2.12}$$

where p_k is the probability of the pattern of stimuli at entry k in the lookup table. In this case, the choice of of 2 for the logarithm base is an arbitrary one as we are interested in comparing the levels of information rather than their absolute values.

In this context, patterns of stimuli which 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 Figures 2.6a and 2.6b), 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 in Section 2.4, 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 self-information on an instant by instant basis. 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.

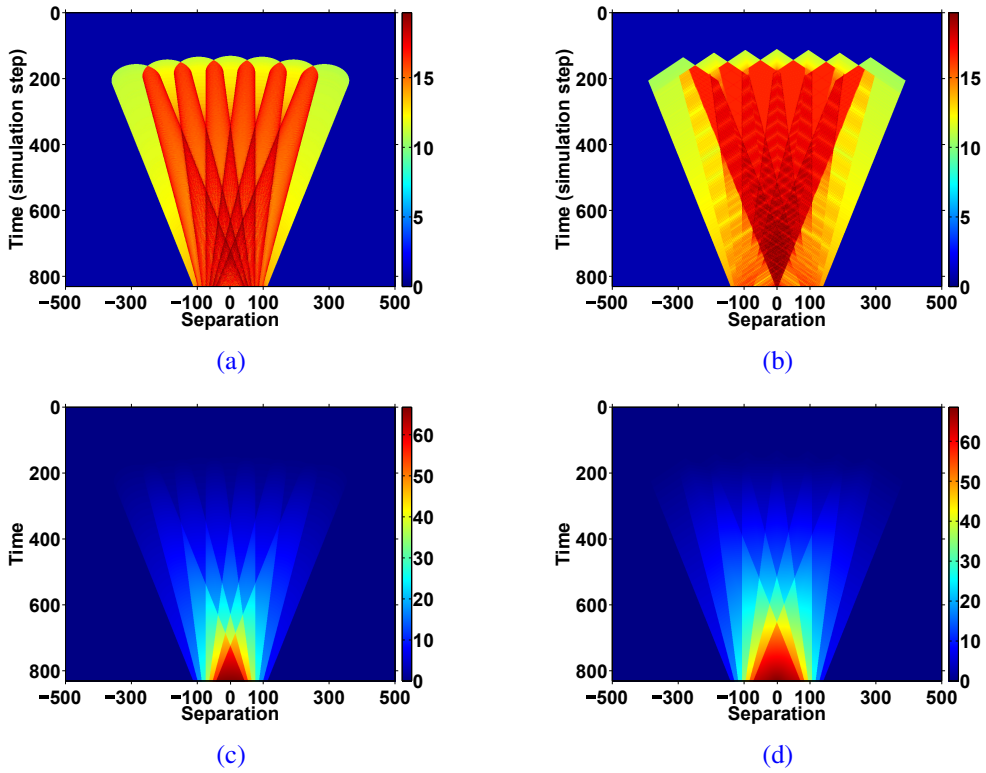


Figure 2.6 Representations of the visual field of the agent. As the object falls at a constant rate over all trials there is an equivalence between its position and time, plotted on the y-axis. The horizontal distance between agent and object is plotted over the x-axis. The same cells, bounded by edges where sensors switch on/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.

Beer (2003) 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 indi-

[1]To clarify, although entropy is the expected value of self-information, we do not consider the entropy of the lookup table 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 the lookup table.

vidual sensors detecting and not detecting the object, as the relative position of object and agent is varied (Figure 2.6). 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 Figures 2.6c and 2.6d, but also that the aforementioned edges have particularly high informational content (Figures 2.6a and 2.6b). 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 2.7). As noted in Section 2.4, the behaviour 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. Trajectories such as that for controller A on catch trials (Figure 2.7a) suggest that the evolutionary process has identified and exploited paths of high information, but it is unclear as 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.

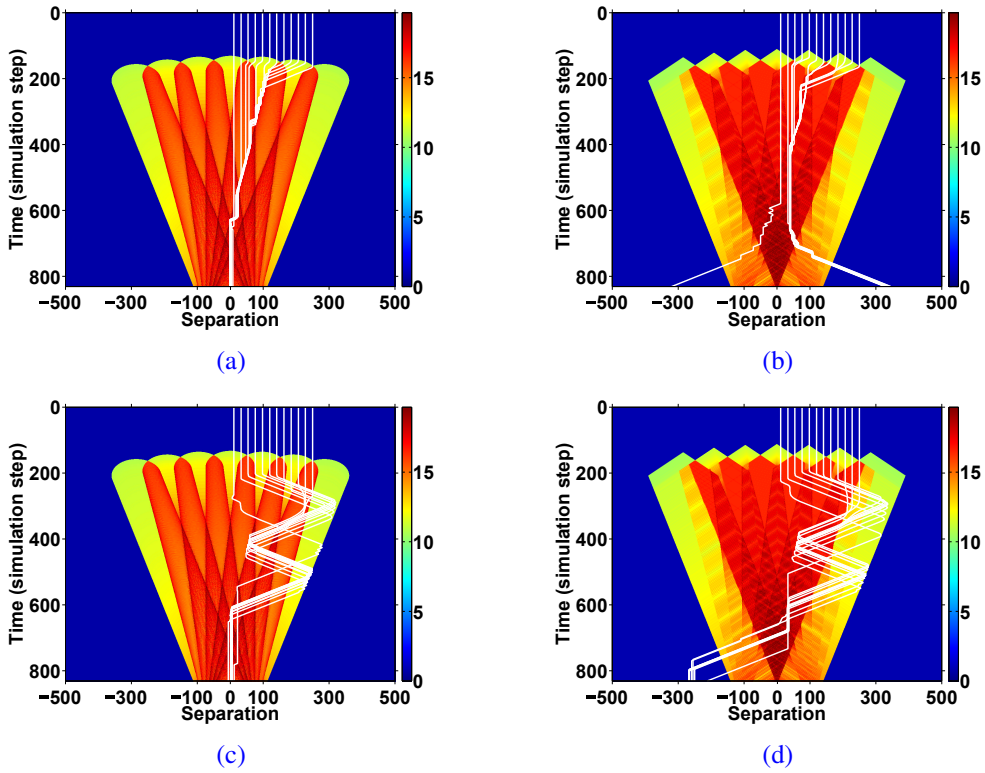


Figure 2.7 Trajectories for controllers A and A1 overlaid on the self-information maps. (a) The trajectories of controller A for catch trials overlaid upon the self-information map. (b) The trajectories of controller A for avoid trials overlaid upon the self-information map. (c) The trajectories of controller 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.

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 used by Klyubin et al. (2005) to measure the capacity of an agent's actuation channel from its motors to its sensors. As we can see in Figures 2.8a and 2.8b 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 maximising information at the evolutionary timescale and at the level of individual controllers. In fact a policy of simply maximising the summed amplitude of stimuli compares well in terms of performance with maximising information (Figures 2.8c and 2.8d). This is also the case when noise is added to the information map and visual field for the two scenarios, respectively (Figure 2.9). Although we have not yet identified it in any controllers, we believe that in general it will be easier for both evolution and individual agents to follow this second policy, but also that it will be more robust at the evolutionary timescale. 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 in the information space, the policy of simply maximising 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 co-evolved with behaviour, 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 2.4.4, performance is more impacted upon by certain sensory disruptions than others, which lends further support to this hypothesis.

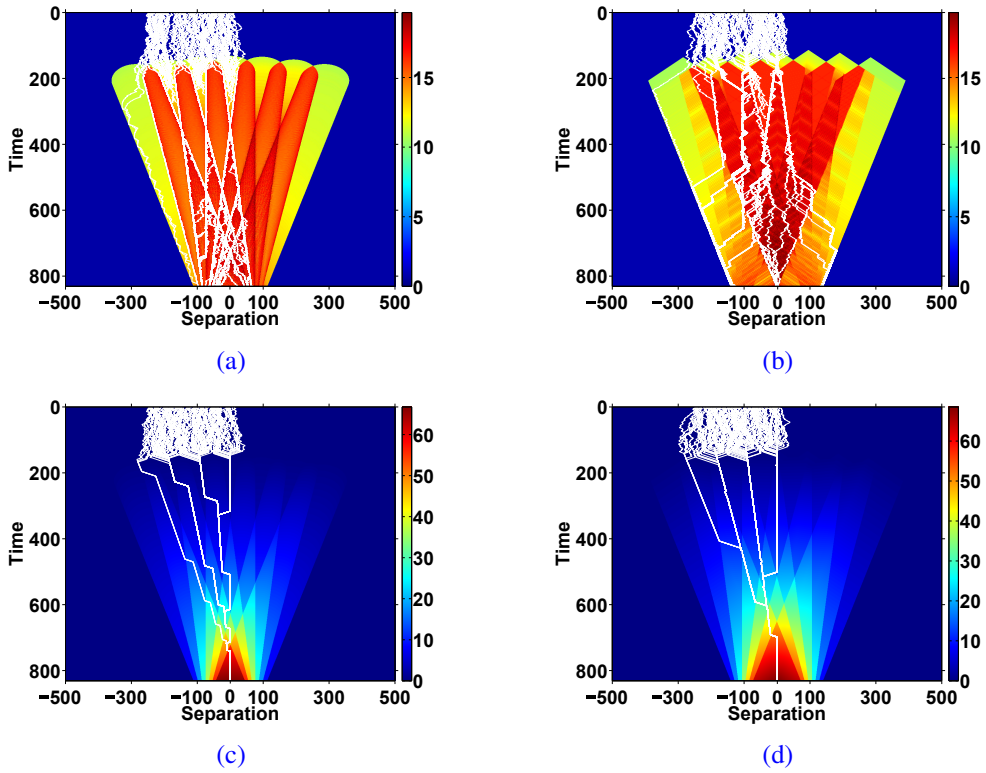


Figure 2.8 Testing simple agent policies. The first policy, shown in (a) and (b), is for the agent to move such as to maximise the self-information appearing to its sensors. The second policy, shown in (c) and (d), is for the agent to move such as to maximise the sum of stimuli intensities appearing 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) Maximising self-information for catch trials. (b) Maximising self-information for avoid trials. (c) Maximising summed stimuli intensity for catch trials. (d) Maximising summed stimuli intensity for avoid trials. Note that maximising stimuli intensity is optimal for catching objects, but shows no distinction between object types. On the other hand maximising information, while suboptimal for both catch and avoid, does lead to distinct behaviours.

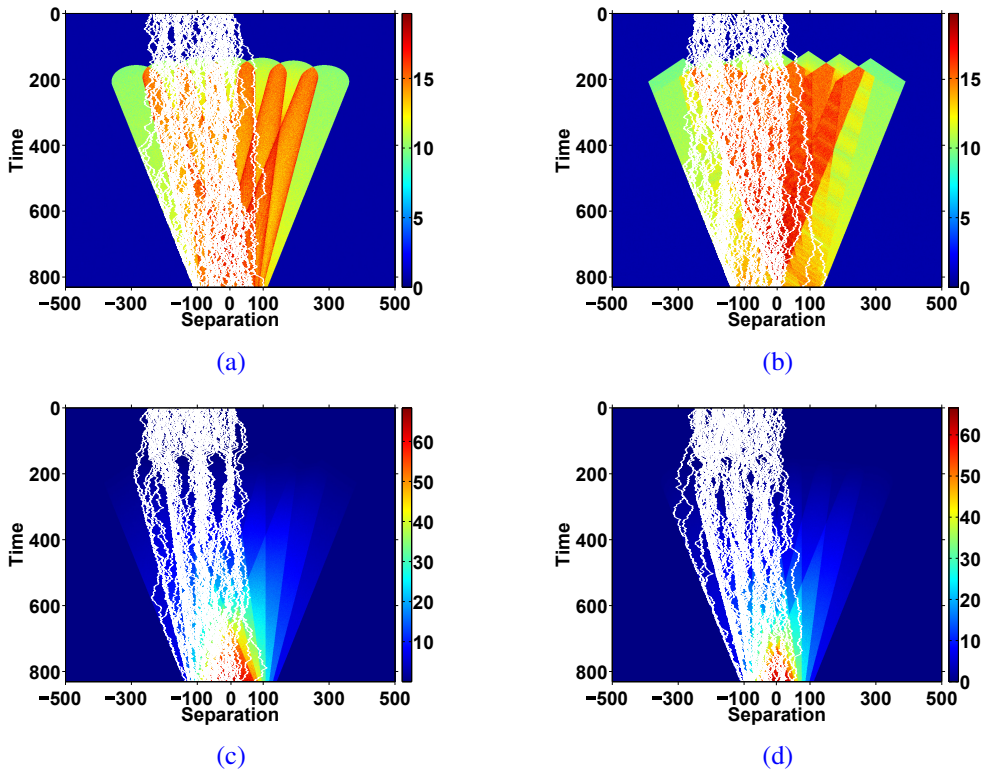


Figure 2.9 The policies of maximising self-information and maximising summed intensity of stimuli are tested under conditions of noisy sensors. The visual fields are subjected to white noise, with a signal to noise ratio of 5:1. (a) Maximising self-information for catch trials under noisy conditions. (b) Maximising self-information for avoid trials under noisy conditions. (c) Maximising summed stimuli intensity for catch trials under noisy conditions. (d) Maximising summed stimuli 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.

2.4.2 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 as it gives clues as to how the controllers perform their functions. In acquiring our measures, we follow a method based on that described by Maass et al. (2003). 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 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 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 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, 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 a capacity for memory in the network. The same test is performed

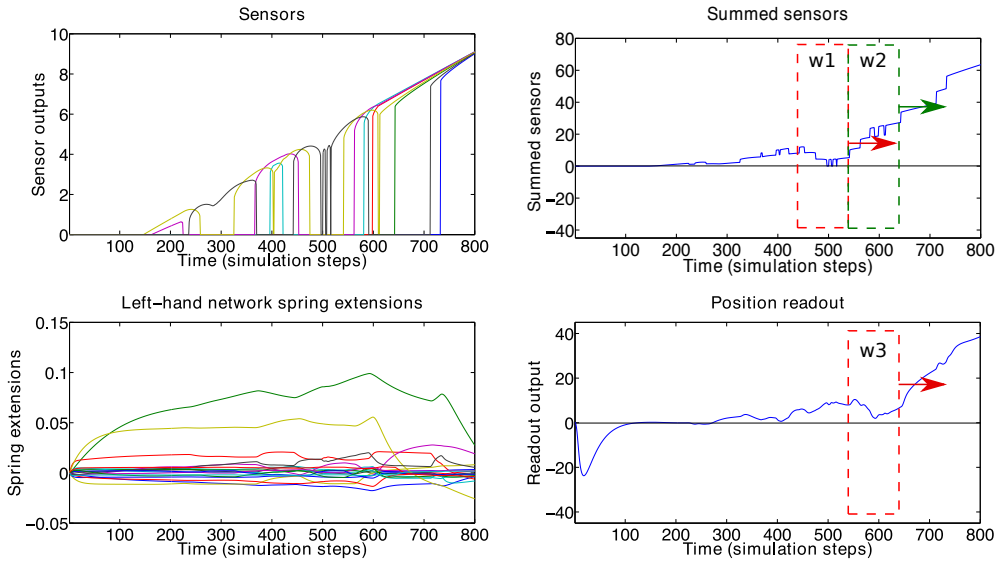


Figure 2.10 Measuring memory capacity. In order to test the capacity of our networks to retain information from the sensory input streams, we begin by creating a target output stream for the linear readout by summing the inputs from all sensors over a single trial (top left panel) into a single time series (top right panel). The readout is then trained to output the target time series, delayed by 10s, which is 100 simulation steps (bottom right panel), from the spring extensions of the network (bottom left panel). We then measure the correlation between the input and output streams over time using sliding windows of 10s duration. As a baseline, we measure the correlation of the target stream against itself 10s ago by comparing the segments in windows w1 and w2. We measure the correlation between target and output by correlating windows w1 and w3.

for readouts which receive only spring extensions as inputs, readouts which receive only spring velocities, and readouts which receive both. Previously we conducted this test with a small set of consecutive segments of the input and output streams (Johnson et al., 2014). 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 seconds, 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 (Figure 2.10).

The results of some of these tests are shown in Figure 2.11. 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 Figures 2.11a and 2.11b 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 2.11b) 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 Figures 2.5a and 2.5b, 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 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 Figures 2.11c and 2.11d, 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 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.

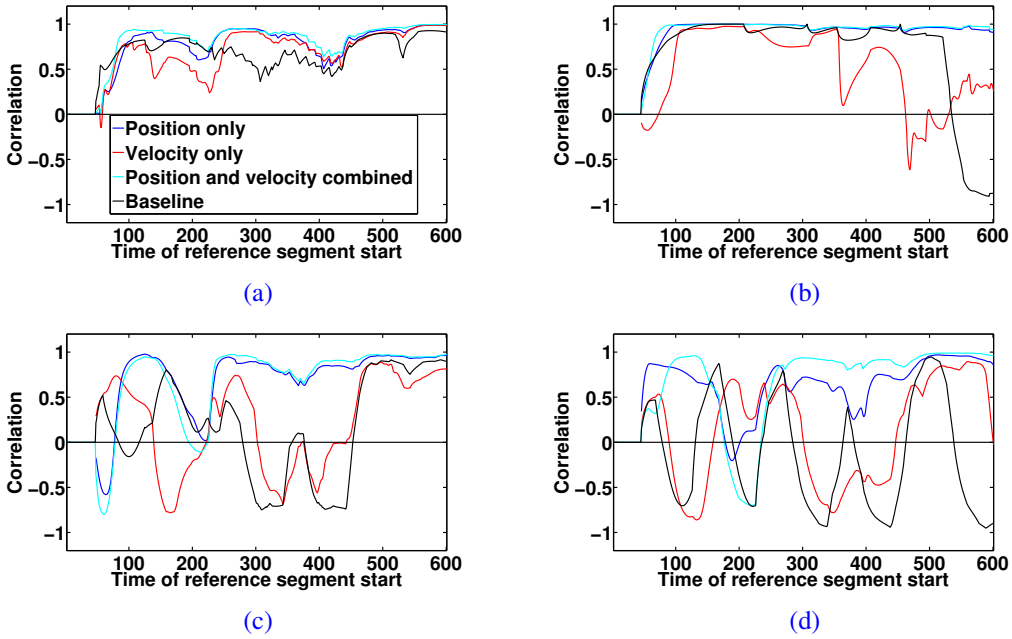


Figure 2.11 Memory capacity measurements. The correlation of the input streams from the two periods is also shown as a baseline. The legend for all four plots is shown in (a). (a) Controller A, trial 5 (catch). (b) Controller A, trial 18 (avoid). (c) Controller A1, trial 5 (catch). (d) Controller A1, trial 18 (avoid). In all plots the baseline trace indicates the zero-memory case and memory is is indicated by positive difference between network outputs and this baseline. Typically the baseline is high and the network memory is therefore only 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.

2.4.3 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 2.12) we used an increment of 0.2 across the whole range, and 0.1 in the range of $[-0.2, 0.2]$ centred about the value already optimised 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, this 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 (Figures 2.5c, 2.5f and 2.5i). As can be seen in Equation (2.7), 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. (2012), the couplings between the MSDs in the network will introduce a certain amount of nonlinearity regardless of which spring model is used.

One discovery which is slightly more surprising is that the genes which encode for the weights

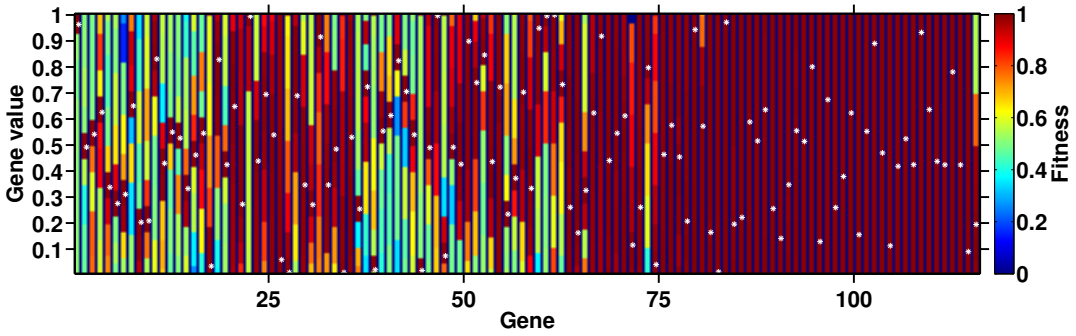


Figure 2.12 Each gene was perturbed one at a time for controller A1. The entire range of values for each gene is explored with a resolution of 0.2, and the interval of [-0.2,2] about the evolved gene value is explored with resolution 0.1. Evolved gene values are marked with white asterisks. For many genes the surface contains one or more plateau, 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.

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 being 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 2.5i, the velocity states will, on average, be low and so the weights applied to them will be of relatively low 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 incorrect velocity encoding will be disruptive of memory effects appearing at the network outputs, and indeed of behaviour 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 more a property which is local to a node rather 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 $[-0.1, 0.1]$ centred about the existing value we discovered that the fitness landscape is in fact not as smooth as it originally appeared (see Figure 2.13b). 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 2.13a). This 1-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.

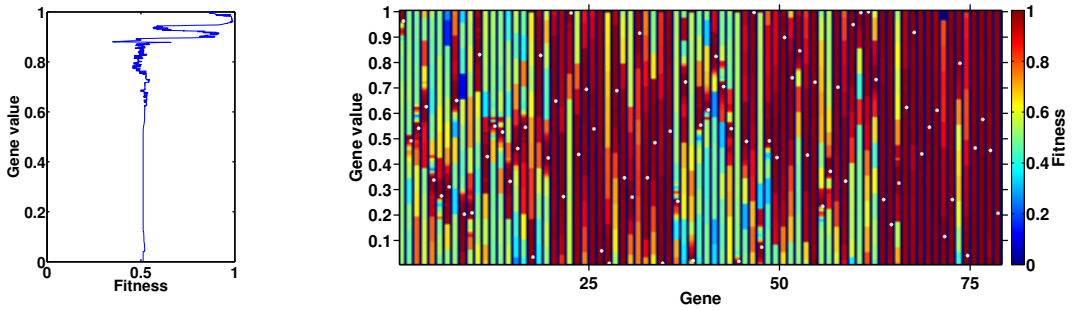


Figure 2.13 (a) The 1-dimensional fitness landscape for gene 1 of controller A1, where all other genes are held at the evolved values. Here it becomes clear that not only are plateaus in the fitness landscape an issue, but that the surface is increasingly noisy in the region surrounding the optimum value. (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.

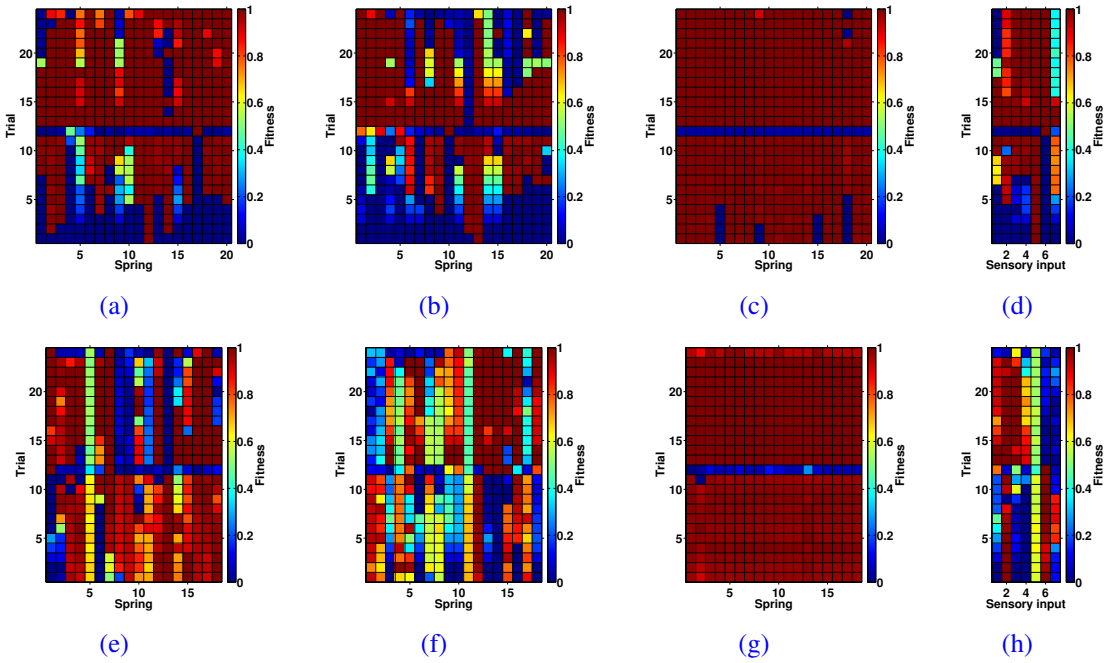


Figure 2.14 Lesion experiments. The top row of plots shows results for controller A and the bottom for controller A1. The rows in plots show the performance on a trial by trial basis. The colour of a grid element shows the fitness of the controller following the lesion. Trials 1 to 12 are the catch trials 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 linearised; one input at a time is disconnected. It can be seen, here as in the perturbation tests, that performance for both networks is relatively robust to the adjustment of many parameters.

2.4.4 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 2.14 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 (Figures 2.14a and 2.14b, 2.14c) 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 but depends on all sensors except sensor 5 for catching (Figure (2.14d).

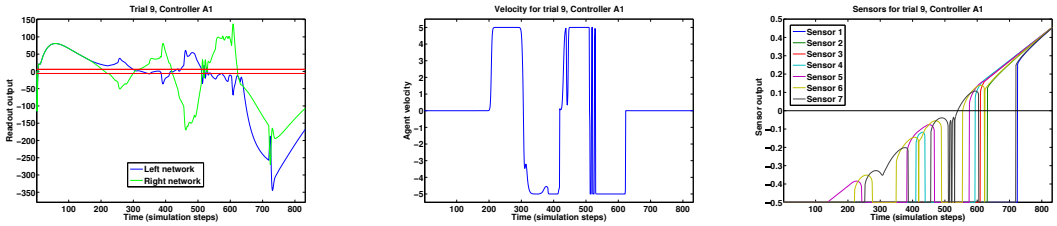
Complete removal of springs from the network causes high failure in both agents, although controller A1 (Figures 2.14e, 2.14f, 2.14g, and 2.14h) is more robust to this than controller A. That this should cause a high failure rate comes as no surprise, given the tightly coupled nature of the network dynamics. Neither controller shows a strong dependence on spring non-linearity; which is in accord with observations in the previous section on perturbation tests for controller A1.

The plots in Figure 2.14 suggest that for both of these agents the most difficult trial is trial 12, 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 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.

2.5 Case studies

In this section we will consider the controllers A and A1 separately, scrutinising their behaviour 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. As these two seem to represent the opposite extremes of the range of behaviours 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.

2.5.1 Controller A1

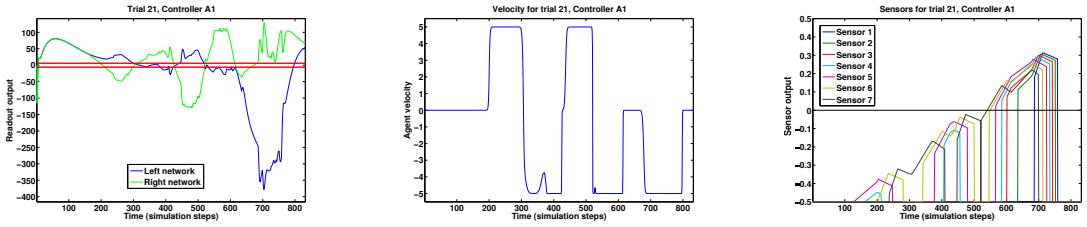


(a) (b) (c)

Figure 2.15 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 normalised and shifted to the interval $[-0.5, 0.5]$, but prior to the delaying effect of the sensory neurons.

As can be seen in Figure 2.15b, 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 2.2). Figure 2.15a shows the network outputs which produced the velocity shown in

Figure 2.15b. 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 non-linear, especially as sensory input tends to rise linearly and show high correlation between not only 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.



(a) (b) (c)

Figure 2.16 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 2.15). (a) Network outputs. (b) Agent velocity. (c) Sensory stimuli.

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.

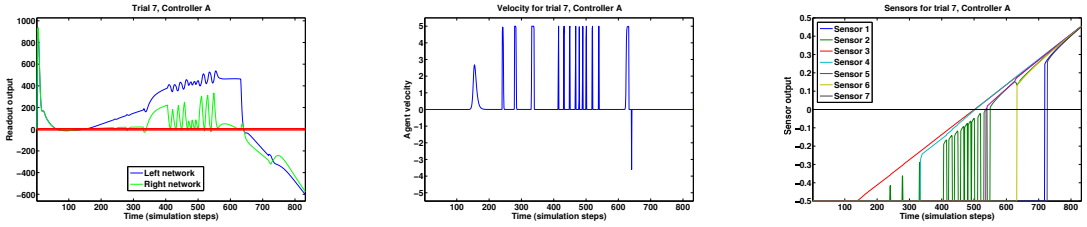
It is often difficult to attribute causality in the sensorimotor loop. For this controller, the trajectories across the information maps shown in Figures 2.7c and 2.7d suggest that this controller is aversive to a lack of information, turning back towards the object at the point where the last sensor switches off. But if we look at $t \approx 300$ in Figures 2.15b and 2.15c, where the agent makes such a turn, we discover that the last sensor 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 which produces a trajectory which appears to respond to edges in the information map by changing direction (Figures 2.7c and 2.7d). 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 behaviour, 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 which 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 which 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 Figures 2.16a and 2.16b, for the corresponding avoid trial, there is some 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 2.16b) and the 'escape' behaviour to be initiated.

2.5.2 Controller A



(a) (b) (c)

Figure 2.17 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 2.7a).

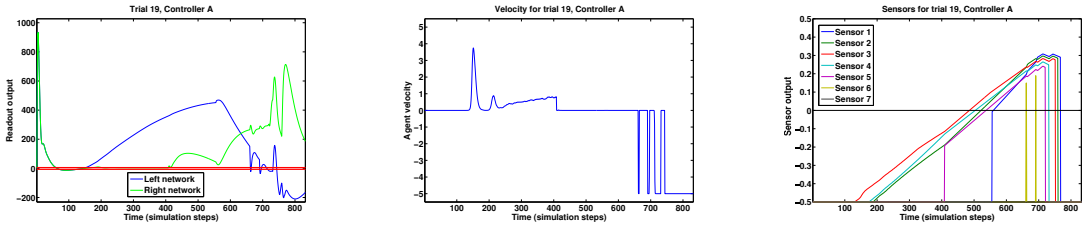
Whereas controller A1 tends to switch both motors, the general pattern for controller A is to keep the left motor at full power until late in the trial and so control its velocity by switching the right motor on and off (Figure 2.17b). In catch trials it inches towards the object by pulsing the right motor. In avoid trials it is still the right motor which effectively controls the velocity, but its action can be smoother and more prolonged than in catch trials (Figure 2.18b).

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 2.17c) 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 one 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 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 Figures 2.17c and 2.17b 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 emphasised the importance of sensors 2 and 6 in centring and catching, respectively, and yet in the lesion tests of Section 2.4.4 it emerged that in the case of trial number 7, the most detrimental lesions were for sensors 3, 4 and 5 (Figure 2.14d). A closer look at Figures 2.17c and 2.17a reveals that although sensor 2 seems to drive motion, it is the combination of its effect and that of sensor 4 which 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 which this agent was seen to follow in Section 2.4.1 and has been discovered to not be a robust solution.



(a) (b) (c)

Figure 2.18 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 2.17). (a) Network outputs. (b) Agent velocity. Note that although there is still a tendency to pulse motors, in the avoid trials velocity occasionally varies more smoothly, as can be seen here between $t \approx 200$ and $t \approx 400$ (c) Sensory stimuli.

In trial 19, the corresponding avoid trial for 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 behaviours 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.

2.5.3 Summary

We have seen that a certain behavioural variety is possible in the networks used here, but as with controllers A and A1, there are various features which all results seem to have in common. Due to the production of large signals in the network readouts, and 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 which switch at appropriate times across all the trials. In many cases, such as that of controller A, this may not be a behaviour which is 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 timescales.

2.6 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 2 simple summing nodes. Hence the body performs the lion's share of the computation involved in producing adaptive behaviour which could be thought of as a reflexive response to different patterns of stimuli. Throughout most of this chapter it has been convenient to describe the two re-

sponses as catch and avoid, but, for example, we could equally conceive of this behaviour 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 behaviour, 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 which 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 behavioural phase 1, where agents position themselves under the falling object, maximising 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 which measures and acts to maximise the information at its inputs, and makes it clear that this is an adaptation at the evolutionary timescale. 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 which 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 (Vogel, 2003), so it would be interesting to adjust the nonlinear terms in Equation (2.7) such 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.

2.7 Conclusion

According to Pfeifer and Bongard (2006), intelligence is "distributed throughout the body", and not solely in the brain. The body of evidence for this is ever-growing, and we believe the work reported on here constitutes a significant addition to that body. We have shown that complaint 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 behaviour with the body as the main computational locus, then it seems reasonable to hypothesise that biological soft bodies could perform a similar function. Furthermore, if this behaviour 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.

In this chapter we presented a novel physical reservoir computing controller which is amenable to 'programming' through the use of an evolutionary algorithm. This is the first instance of physical reservoirs engaging directly with the environment via exteroceptive sensors, and the first time that they have been required to select between behaviours in a scenario which is analogous to the fight or flight response (Cannon, 1953). In the following chapter we continue in the same vein, but with a more challenging problem for the controllers to solve.

Chapter 3

Stretching fading memory

In this chapter, we continue the line of enquiry that we began in Chapter 2. Having discovered that the decision-making task there could be achieved with little or no use of the fading memory which is inherent in MSD networks, we next selected an experiment for testing our MSD network controllers that explicitly requires memory. In addition, where our earlier controllers controlled motion in a single dimension, the task we assigned them here was to control a robot moving in a maze in a two-dimensional plane. Also, to make the task more challenging, this time we added sensor and motor noise to our simulation, as well as some randomisation of the dimensions of the maze. Ultimately, we had some successes but found it so difficult to obtain successful controllers for this task that we concluded that our controller design is not optimal for evolving behaviours which require memory over an extended period. In the last part of this chapter, we introduce a novel bistable MSD network, inspired by this problem.

3.1 Maze navigation and memory

In this section, we describe the controller design, algorithms and parameters used in obtaining our most successful controller. On average, this controller performs correctly and reaches the target location in approximately 90% of trials. In the remainder, it typically turns the wrong way at the junction, but it also occasionally crashes. We consider this our best controller because it is our only controller to achieve such a high score and be robust to both sensor and motor noise, and also to variation in the dimensions of the maze.

3.1.1 Methods

3.1.1.1 The experiment

We based our experiment on one introduced by Blynel and Floreano (2003), which requires navigation in a T-maze and the use of memory to select an appropriate action. For every trial, the robot is started from a position at the bottom of the lower corridor (Figure 3.1). A target zone, marked as a black square on the ground, is placed on either the right or the left end of the upper corridor. The first time the robot is started, its controller should be initialised with no pre-existing internal states, and it should make its way to the target zone and halt there. The robot is then teleported, as it were, back to its initial position with the internal state of its controller preserved, and for the next trial it should remember which way to turn at the junction so that it returns directly to the location of the target zone.

The requirements for solving this problem are clear. First, there is the task of finding the target zone the first time the robot runs. This is a straightforward navigation problem. The robot must drive up the lower corridor towards the junction, and then turn either left or right to reach one of the ends of the upper corridor. When the corridor end is reached, the robot should either stop, if the target is there, or turn around and drive to find the target at the other end of the corridor. For

the next part of the problem, somewhere in the controller there must be a state or combination of states which encodes which end of the upper corridor the target zone was found in, and which will determine the direction the robot turns the next time it is returned to the start position. While we can talk about these requirements separately, and even evolve them in sequence, as we explain below, we do not mean to imply that our controllers with deal with them separately. In our work with small MSD networks we have found that the different aspects of control actually appear to be superimposed upon each other.

Blynel and Floreano simulated a Khepera robot controlled by a continuous time recurrent network (CTRNN) for their experiment. We used a simulation of an E-Puck robot, which has a similar design and sensor configuration to the Khepera, controlled by two pairs of MSD networks. For the simulation of the E-Puck robot and the maze, we used the Enki 2D robot simulator C++ libraries (Magnenat et al., 2013).

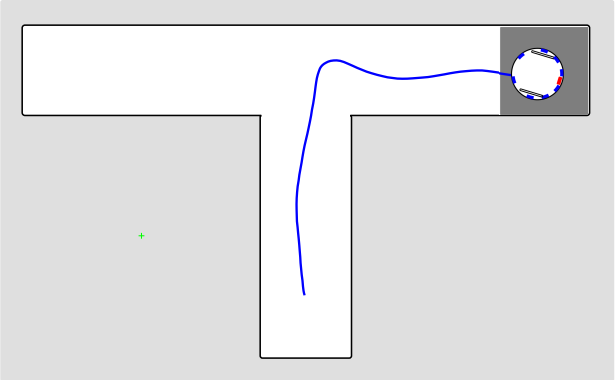


Figure 3.1 The E-Puck robot in the T-maze. The target zone, indicated as a dark grey square, can be placed on either side of the maze, but the robot always starts from the same place near the bottom of the lower corridor, subject to small variation in its position and orientation.

3.1.1.2 Simulation

Our MSD networks were simulated in the MATLAB IDE as described in Chapter 2. All other controller aspects were also simulated in MATLAB as in Chapter 2 except where specified otherwise. For simulating the robot and the maze walls, we integrated the Enki library into our simulation using MATLAB mex functions. The Enki simulation is run with a simulation step of $0.1s$. As the E-Puck does not include a ground sensor, we added that sensor and the reward zone to the MATLAB part of the simulation. The robot is $7.4cm$ in diameter, and the maze corridors are $14.8cm$ wide. The nominal outside dimensions of the maze are $92.5cm$ by $53.65cm$.

3.1.1.3 Controller design

In the experiment of Chapter 2, we were successful in using a bilaterally symmetric pair of mass-spring-damper networks to control an antagonistic motor pair for motion along a single dimension. For the control of our simulated robot moving in a two-dimensional maze environment, we chose to use an antagonistic network and motor pair for the control of each wheel (Figure 3.2). In this case, we also chose to break the symmetry of the controller. For behaviours like maze navigation, a completely symmetric controller is problematic as a robot can get trapped at junctions and corners, for example, making small turns to the left and right repeatedly. For this reason, we only make the networks that control the backwards motion of the motors a symmetric pair. For this pair, the networks have identical morphologies, have the sensors connected in reverse orders, and have the same evolved readout weights, as we described for our controller in Chapter 2. For the front

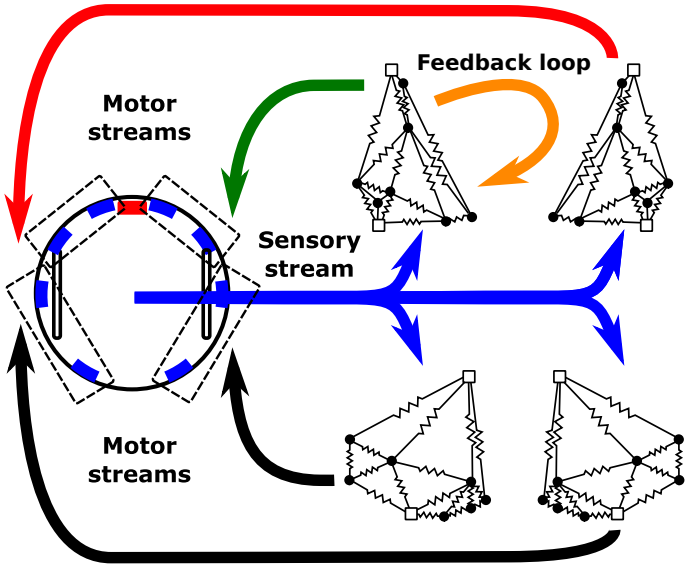


Figure 3.2 The E-Puck robot and its controller. Camera (unused here) shown in red, on the front of the robot. Infrared (IR) distance sensors shown in blue. The rectangles with dashed outlines show how the sensors are grouped in pairs. Stimuli from paired sensors are summed before being input to the networks, so that the networks receive 5 inputs: 4 from IR sensors and one from the ground sensor, located under the centre of the robot, which detects the target zone. The robot's wheels are shown as rounded open rectangles. As indicated, we implemented a feedback loop on the front left MSD network only.

network pair, we follow the same principles except that we evolve the readout weights for the two networks separately, so that they have the same internal dynamics but distinct outputs. As before, each network readout output is passed to a sigmoidal motor neuron. The output of each motor neuron is multiplied by a factor of 10, so that the robot's wheels can run at velocities in the interval $[-10cm \cdot s^{-}1, 10cm \cdot s^{-}1]$.

All MSD networks have 9 nodes, as in the controllers of Chapter 2. As before, we randomly generate all MSD networks at the beginning of an evolutionary run, and they remain fixed throughout the run. In order to reduce the dimensionality of the sensory input to the networks, we grouped the E-Pucks 8 infrared sensors into pairs, so that each network receives 5 inputs, 4 from the infrared sensors and one from the ground sensor (Figure 3.2). Sensor neurons, implemented as in Chapter 2, filter the sensory streams before they are input to the networks. As there were only 5 sensor inputs to the networks but 7 free nodes in the networks, at the beginning of evolution 5 of the free nodes were selected as input nodes. In our initial design each input node received an input from a single sensor, but we later modified this so that each input node received a weighted sum of all sensory inputs.

We implemented controller feedback as described in Chapter 2, on the front left network only. We chose to apply feedback on only one network, partly for the aforementioned purpose of breaking symmetry in the controller, and partly to avoid increasing the evolutionary search space any more than necessary with additional parameters. Feedback was applied to all 7 free network nodes, and the feedback for each node was generated using a separate linear readout to combine the spring extensions for this network. All feedback weights were evolved. In all the successful and partially successful controllers we observed, feedback did not appear to perform any obvious memory function, but rather seemed to stabilise control to reduce the probability of the robot crashing during turns at the junction. However, our controllers were ultimately black boxes without separable control mechanisms and we were unable to clearly identify any individual aspects of the controller in the high-dimensional state space formed by the group of MSD networks.

3.1.1.4 Evolving controllers

3.1.1.4.1 Guiding evolution When we tried to evolve robust controllers in a single evolutionary run, we obtained no good results. We eventually hit upon a number of constraints which we applied in order to guide evolution. The first of our constraints was that we chose to make the robot turn right by default, except in memory cases. A second was that we set the duration of trials such that a partially successful robot tended not have time to wander far from its destination before the clock ran out. For the trial where there was no memory and the target zone was on the left hand side of the maze, we allowed 400 seconds for the robot to drive first right and then left. For all other trials, the robot was expected to drive directly to the target, and so we allowed only 90 seconds. We also took an incremental approach to evolution, employing two or three stages. Our algorithms minimise cost, which is always in the interval [0, 1]. Where it seems more natural we refer to fitness rather than cost. Fitness is the opposite of cost, equal to $1 - cost$.

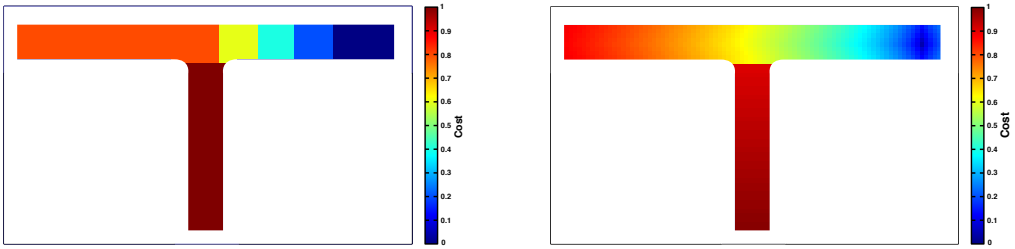
3.1.1.4.2 Evolution - stage one As we observed earlier, the first requirement for the controller is to navigate the maze without crashing and find the target zone. Therefore, we evolved this capability first. There were two trials in this stage, although they were repeated multiple times, for reasons explained in Section 3.1.1.4.5. In the first trial, the target was placed on the right-hand side of the maze, and the robot was required to drive directly to it. In the second trial, the target was placed on the other side of the maze, and so the robot was expected to turn right initially and then turn around and drive to the left-hand side. We used a highly proscriptive cost function in this stage (Figure 3.3a). Using this cost function, controllers which fail to take the robot into the top corridor of the maze are assigned the highest possible cost, of 1.0. In the case where the trial requires the robot to reach the target at the right-hand end of the top corridor, the cost is reduced by 0.2 for any controller which reaches the top corridor, and then by a further 0.2 for each successive zone which takes the robot closer to the target zone. For the trial which requires the robot to first drive to the right-hand corridor end, and then turn and go to find the target at the left-hand end, the situation is modified; a controller will only be rewarded for driving into the corresponding zones on the left-hand side if it has first driven towards the right.

For this stage we found that a genetic algorithm was prone to premature convergence to sub-optimal solutions, and so we used a macroevolutionary algorithm (MA), as described in Chapter 2, with a population of 400 candidates. Once we obtained a controller which succeeded in this stage, we halted evolution and continued to stage two.

3.1.1.4.3 Evolution - stage two In the second stage, we used the same population from stage one and restarted evolution to complete the required behaviour. This time, we had four trials: 1. drive right without memory, 2. drive right with memory, 3. drive left without memory, 4. drive left with memory. The cost function used in this stage (Figure 3.3b) also only rewards controllers which make it to the top corridor fo the maze, but for any robot which ends the trial in that corridor, the cost is a simple function of distance from the target zone.

For this stage, we switched from using the MA to using a distributed genetic algorithm (DGA), (Collins and Jefferson, 1991; Husbands, 1994) with a population placed on a toroidal grid and local neighbourhoods used for selection and mating. At this stage, it seems that the faster convergence of the DGA compared to the MA may be advantageous. As memory is being evolved 'on top of' navigation, any reachable complete solutions are likely to be placed close in parameter space to solutions for navigation capabilities. We used a rank roulette selection algorithm, with candidates only competing locally in a square neighbourhood of 9 candidates. We did not employ crossover; for each competition, the candidate selected for reproduction replaced the candidate selected for removal and was then mutated. For this stage, we used a high mutation probability, of 0.1 for each gene. For every mutation, a gene would be mutated by plus or minus 10% with probability 0.1, and otherwise be mutated to any possible value.

3.1.1.4.4 Evolution - stage three We found that in some cases we were able to refine a controller and improve its performance by applying a hill-climbing algorithm. In this case, the algorithm operates on a population of one genotype. The genotype is copied, mutated and evaluated. If the copy performs better than the original, then the original is replaced with the copy, otherwise the copy is discarded. For our best controller, this stage was not required.



(a)The cost function for stage 1 rewards controllers according to how many zones they pass through which lie on the route to the target zone.

(b)The cost function for stage 2 rewards controllers based on how close they are to the target zone at the end of the trial.

Figure 3.3 Cost functions.

3.1.1.4.5 Variation We found that it was relatively easy to obtain controllers which could solve the given problem when our simulation contained no sources of noise or variation in the environment. However, we discovered that these controllers were not robust to the addition of sensor or motor noise, or to variation in the dimensions of the maze or even variation in the length of time allowed for each trial. We were also unsuccessful when we tried to continue evolving these controllers to cope with the addition of variation. From this we conclude that evolution was taking a cheap option to achieving the behaviour, and relying on a very specific combination of controller states and environmental features. In other words, the robot was following the same paths for every set of trials, and not actually capturing in memory which side of the maze the target was on. The only way we could achieve the evolution of robust controllers was to build variation into the simulation from the very beginning of evolution. To sensor and motor signals, we added white noise at the level of ±5% of the maximum signal. We also introduced some small variation in the lengths of the maze corridors, and in the initial position and orientation of the robot. In order to get an accurate fitness for a controller, we repeated the complete set of trials 10 times and took the average of the fitness for each set.

3.1.2 Summary

While our controllers for the robot in the maze were not as successful, in terms of evolving a number of robust solutions, as the controllers in Chapter 2, we were still encouraged to obtain one reliable controller. The difference in success levels indicates that this is, as expected, a significantly more challenging task for controllers to achieve than that in our earlier experiment. We halted our efforts on this experiment due to a decision that enough time had already been dedicated to it, but speculate that with more time, and perhaps larger networks, an effective MSD network controller design could be found.

The addition of a memory requirement to the task is clearly one of the main reasons that the task is so challenging. While considering this issue and potential solutions, we developed a simple bistable MSD network which is capable of persistent memory, and in principle could be incorporated into larger MSD network controllers for controlling the behaviour of robots and virtual creatures.

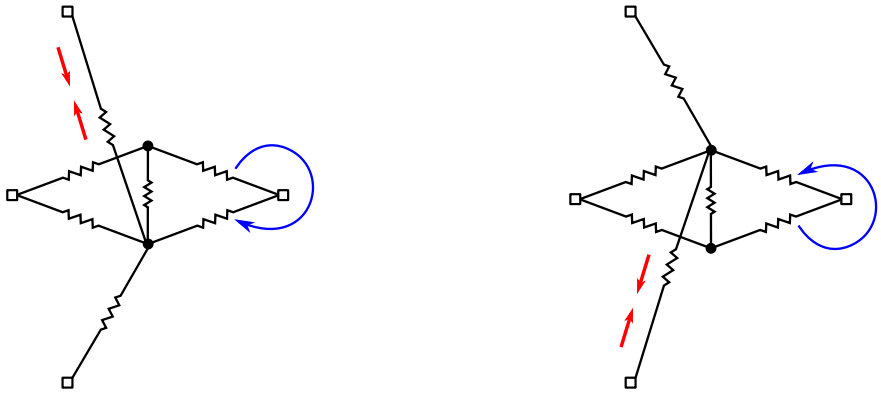


Figure 3.4 A bistable MSD network. Fixed nodes are drawn as squares, free nodes as circles. If enough contractile force is applied to the relevant spring (red arrows), the diamond-shaped sub-network can be flipped horizontally (blue arrow).

3.2 Bistability

Hauser et al. (2012) have already shown that the fading memory limitation in MSD networks can be overcome through the use of a feedback loop. In this section, we show that, with small modifications, a network can latch information, without the need for a feedback loop or input to maintain its state. We demonstrate this with a simple example.

The first modification we make is to introduce additional fixed nodes (Figure 3.4). As usual, we hold the network in place by fixing the leftmost and rightmost nodes, but add additional fixed nodes above and below the network. We connect a single mass-spring damper from each of these fixed nodes to the same node in the main network. By actuating these new elements we can then manipulate the network to switch between two stable states.

We used Google's *liquidfun* (Google, 2013) a two-dimensional physics engine library in the C++ language, which extends *Box2D*, to construct our simulation. We use this physics engine for all of the work described in part two of this thesis. The network is constructed out of circular bodies, which are connected by distance joints which can behave like damped springs. The parameters we used were found by trial and error and were the first set that gave stability, but have no significance beyond that. The network size is 4 by 8 metres. Nodes are circular bodies with radius $0.1m$ and mass of approximately $3140kg$. All distance joints have a natural frequency of $1Hz$ and a damping coefficient of 1.

The second difference between this network and those of Hauser et al., as well as all of our other simulations in this part of the thesis, is that the magnitude of the input to the system must be much greater. We found that if we applied a sufficient force (which varied in this simulation, but was in the order of $1x10^6N$ over $2s$) the diamond shaped configuration in the centre of the network can be flipped across the horizontal axis and will stabilise either way up.

If a linear readout was connected to this network, then as long as the weights for the active springs were far enough apart its output could reflect which of the two stable configurations, or fixed-point attractors, the network would return to in the absence of input. This could cause a mode switch in the input to output mapping of the network. As part of a more complex network, such a mechanism could conceivably be used to solve the problem of the T-maze which we saw in Section 3.1. In the example we give here, we use a very simple network, but this need not be the case. It could be that a more complex network is flipped in its entirety, that only part of a larger network is flipped, or even that multiple parts of a network may be flipped independently.

Conclusion

In this chapter, we have looked at two possible forms of extended memory in MSD networks. The first is memory over a finite but extended period, where a controller must keep track of which way to turn for as long as it takes the robot to reach the maze junction. The second is a persistent form of bistable memory, where the network is manipulated to switch between stable attractors.

The fact that we only obtained a single generally robust controller for the T-maze memory behaviour is disappointing, but the result is still significant, as it shows that it is possible for controllers incorporating small MSD networks to solve this kind of problem. We tried many controller designs, and many variants of our approach to evolving them before we called time on this experiment, but for all we know we could be very close to a design and process which could lead to repeatable success. In echo state networks (Jaeger, 2002), the capacity for memory can be tuned by adjustment of the spectral radius. It may be that a similar method could be applied to the parameters of our MSD networks to improve the probability of evolving successful controllers, but we suspect that the biggest problem here is related to a different issue: the fact that sensors are driving 5 out of 7 free nodes in each network. In order to achieve the navigation part of the task, the controller must be evolved for strong responses to sensory input, but it may be that in such small networks these strong responses tend to dominate over the long-term transients required for extended memory.

In Chapter 2 we showed that MSD network controllers are capable of active perception and decision making. In this Chapter we added both complexity to the behaviour that was to be controlled and a requirement for added robustness in the controller. As well as implementing memory, our successful controller was able to deal with navigation in two-dimensions, where previously we had only one. This result is important to our thesis, as in the following chapters we consider the potential for virtual creatures in two-dimensional environments with MSD network bodies, which could be incorporated into their own controllers as reservoirs.

Chapter 4

Internalising actuation

This short chapter forms a bridge between the two parts of the overall thesis, by connecting ideas from morphological computation and MSD networks to virtual creatures (Sims, 1994). In part two of the thesis, we continue to work with MSD networks, but rather than using them as controllers which may be considered separately from the bodies they control, we move to using them to model the bodies of swimming virtual creatures. A crucial change required by this move is to modify the mode of actuation of the networks from external to internal. Here we show that this change need not have a deleterious effect on computing power.

4.1 Introduction

Previously, we looked at the kinds of uses MSD networks might be put to as controllers, but ultimately MSD networks are of interest because they can be used to model the bodies of animals and robots (Hauser et al., 2011). Our networks, like those of Hauser et al., were passive soft structures held in place by two fixed nodes, and actuated by external forces. For a computational device which maps from an input stream to an output stream, this is an acceptable state of affairs, but we are interested in bodies which are free to move under their own steam, so to speak. A critical aspect of this is for the body to be actuated from the inside, and not a passive structure which is driven from the exterior.

In this chapter we revisit one of the problems Hauser et al. used to demonstrate the power of MSD networks, the emulation of a Volterra series operator, but with the modification of driving the networks internally by actuating the MSDs themselves. Nakajima et al. (2013, 2015) have shown that simulated and robotic octopus arms can be used as reservoirs for similar tasks, but their arms were also essentially passive structures, driven by motors at the arm's base. On the other hand, authors including Schramm and Sendhoff (2011) and Joachimczak et al. (2016) have simulated swimming *animats* (Wilson, 1991) or virtual creatures with internally actuated MSD network bodies, but with a focus on behavioural outputs, rather than morphological computation or coordination. Although we are also ultimately concerned with behaviour, we aim to draw these two strands together, with MSD network bodies which are internally actuated, involved in interesting behaviour, and functioning as reservoirs. This is our first step in that direction.

4.2 Methods

4.2.1 The task

The problem to be solved is for the linear readout to be trained such that the reservoir computing system emulates a Volterra series operator (Figure 4.2b), which describes a nonlinear, time-invariant system with fading memory, defined by the following equation:

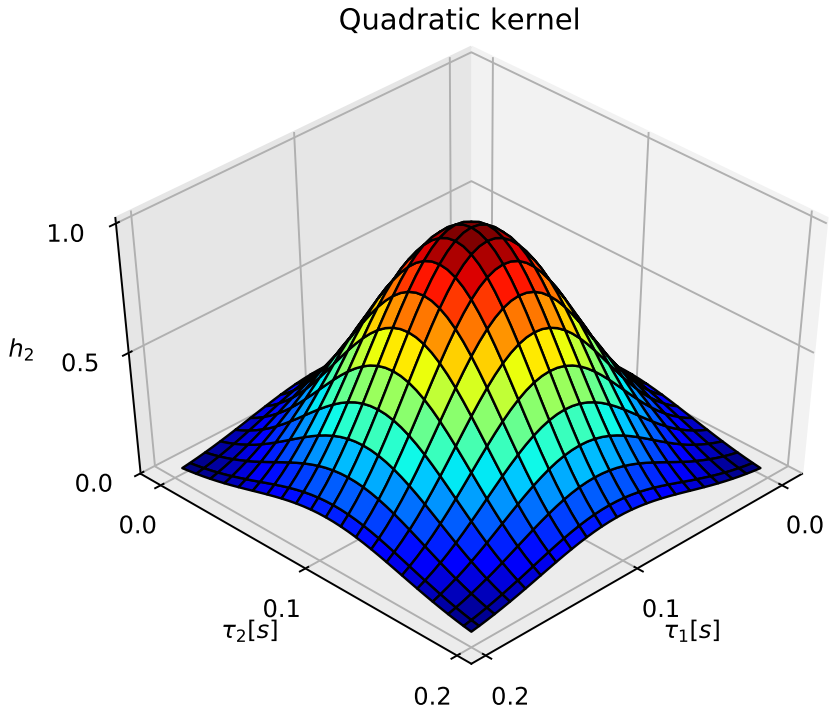


Figure 4.1 The Gaussian kernel used in the Volterra series operator.

$$y(t) = \mathcal{V}u(t) = \int \int_{\tau_1,\tau_2 \in \mathbb{R}_0^+} h_2(\tau_1, \tau_2) u(t - \tau_1) u(t - \tau_2) d\tau_1 d\tau_2 \tag{4.1}$$

The kernel, h_2, is a Gaussian given by

$$h_2(\tau_1, \tau_2) = exp((\tau_1 - \mu_1)^2/2\sigma_1^2 + (\tau_2 - \mu_2)^2/2\sigma_2^2) \tag{4.2}$$

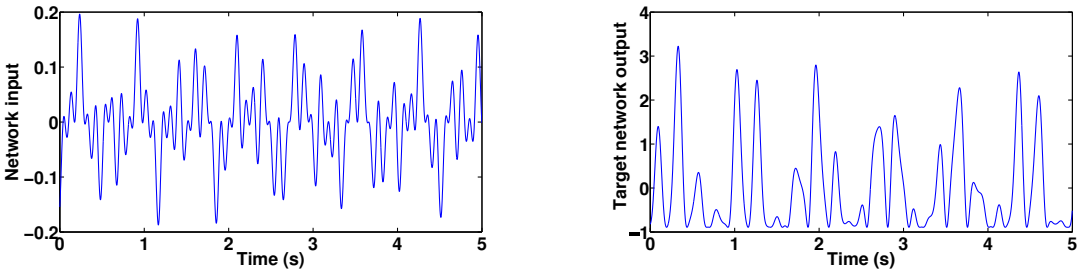
with $\mu_1 = \mu_2 = 0.1, \sigma_1 = \sigma_2 = 0.05\,(in\ seconds)$, defined for $\tau_1, \tau_2 \in [0, 0.2]s$ (Figure 4.1). As pointed out by Hauser et al., the operator incorporates temporal integration, over the delays τ_1 and τ_2, and nonlinearity in the quadratic term $u(t - \tau_1)u(t - \tau_2)$. In other words, for non-trivial input streams this is a challenging function to compute. The input stream used here, $u(t)$ (Figure 4.2a), is a product of three sinusoidal functions with different frequencies

$$u(t) = sin(2\pi f_1 t) \cdot sin(2\pi f_2 t) \cdot sin(2\pi f_3 t) \tag{4.3}$$

with $f_1 = 2.11$, $f_2 = 3.73$, $f_3 = 4.33Hz$, resulting in a signal with a period of $100s$.

4.3 The tests

We set out to answer three questions. First, can a physical reservoir computing system using an internally actuated MSD network perform comparably with one which uses an externally actuated one? That question was answered affirmatively, and so two additional questions followed. For



(a)The input stream (Equation (4.3)), a product of three sinusoidal functions, with a resultant period of $100s$.

(b)The target output stream (Equation (4.1)), which implements a Volterra series operator.

Figure 4.2 The input and target output streams for the Volterra series operator.

a system with an internally actuated network, how does performance degrade when we reduce the size of the reservoir? And finally, for a given reservoir size, how is performance affected by varying the number of elements used as inputs to the system?

4.4 Simulation

So that we could compare our results directly to those of Hauser et al. (2011), we modified Hauser's own program code for simulating and training mass-spring-damper networks, available from https://github.com/helmuthauser/Matlab_Morphological_Computation_Simulation. The implementation is essentially as we described in Chapter 2, except that this code uses the lengths of springs in the readouts, whereas we previously used the extensions of the springs.

For each spring which is used as an input to the network, we add an actuator to the simulation step by simply adding the instantaneous value of the input stream, without applying any weights, to the force generated in the spring and its attached damper due to its extension and rate of extension respectively.

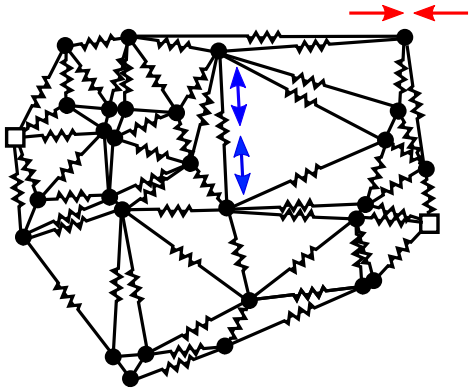


Figure 4.3 An example of the MSD networks used in this chapter. The red arrows indicate the original mode of input to the network, as in (Hauser et al., 2011), where network nodes are driven back and forth along a single axis. The blue arrows indicate our method, where springs are directly driven by internal actuation.

4.5 Results

Hauser et al. used a network with 30 nodes and a proportion of $1/5$ nodes as inputs. As the node positions are selected randomly, the resultant Delaunay triangulation will have a variable number of edges, and so the number of MSDs in the network varies, but for a 30 node network, we have

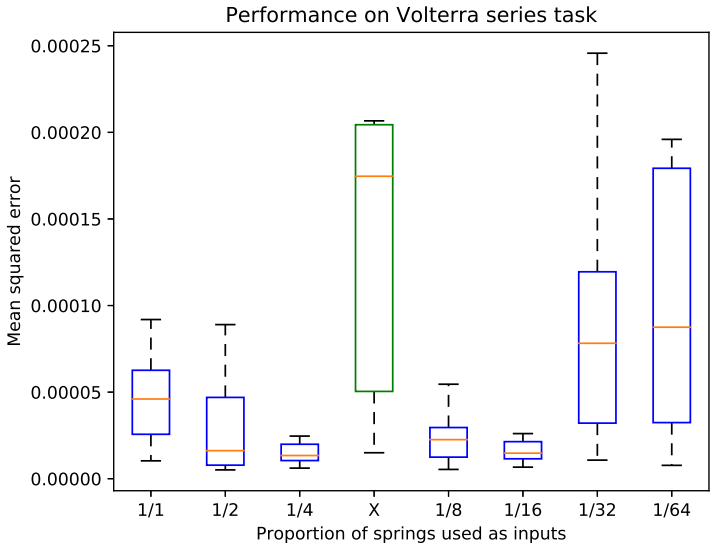


Figure 4.4 Performance for a system with a 30 node network. We varied the proportion of network springs used as inputs, and recorded the mean squared error for 10 networks with each performance. We also show the results for an externally actuated network, marked 'x' and drawn green. Outliers are omitted from the plots.

always seen approximately 80 MSDs. As a baseline for comparison, we ran the Volterra series learning task 10 times with this configuration and recorded the mean square error between the target and actual system output streams.

When we modified the network input mode, we varied the proportion of springs that we used as inputs from using every spring as an input, to using every other spring, and so on up to using one spring in 64. This last proportion results in only a single spring being used as an input. We discovered that for this size network, performance was actually improved over that of the externally actuated network (Figure 4.4), in all cases. What we also see is that at the extremes of our parameter sweep, where we use only a single spring or all springs as inputs, the mean square error increases slightly over the best cases, which use proportions between 1/4 and 1/16. We followed the same steps using smaller network sizes, with 20, 15 and then 10 nodes (Figure 4.5). Performance gradually drops off as the size is reduced, but is less dependent on the proportion of springs used as inputs to the network. When we looked at the degree to which activity was correlated across the networks for these extreme cases (Figures 4.7 and 4.8 illustrate these cases, while Figure 4.6 illustrates the baseline conditions for comparison), we see that, as expected, the network which is driven by all springs shows a high degree of correlated activity. Surprisingly, this had a smaller impact on performance than expected, with the network achieving a mean squared error of only 1.7379e-4, far from the best result we have seen, but still a relatively small error.

Discussion

The results of this experiment are encouraging in all areas. First, we modified the method of inputting to the network from external to internal actuation, and saw comparable performance. This is of interest because animals and robots are internally actuated. As already demonstrated in robots by Zhao et al. (2013), Caluwaerts et al. (2014) and Eder et al. (2017), robot bodies may be used as reservoirs of proprioception, which may be combined using simple linear readouts into motor control signals for simple behaviours. Here we confirm that virtual creatures with internally actuated MSD network bodies will also be capable of complex processing requiring nonlinear

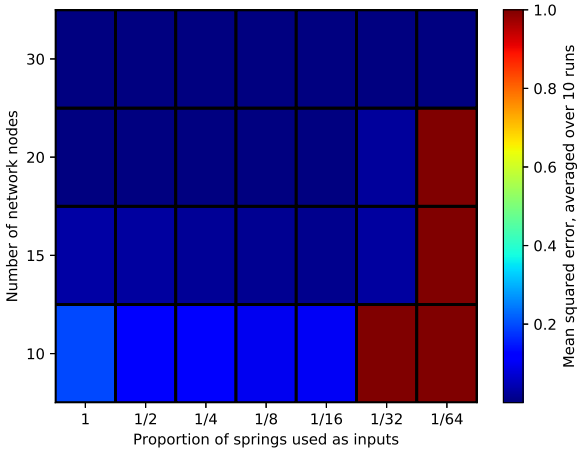


Figure 4.5 The effect on performance of varying the two parameters of interest for our modified system. The first parameter we vary is the number of nodes in the network. The second varied parameter is the proportion of springs that we use as input actuators. For each parameter pair, we ran the simulation with 10 randomised networks, and averaged the mean square error. The entries which show complete failure (in red) are due to the input proportion being so low that no springs are selected as inputs. Performance gradually drops off as the network size is reduced, but shows less dependence on the proportion of springs used.

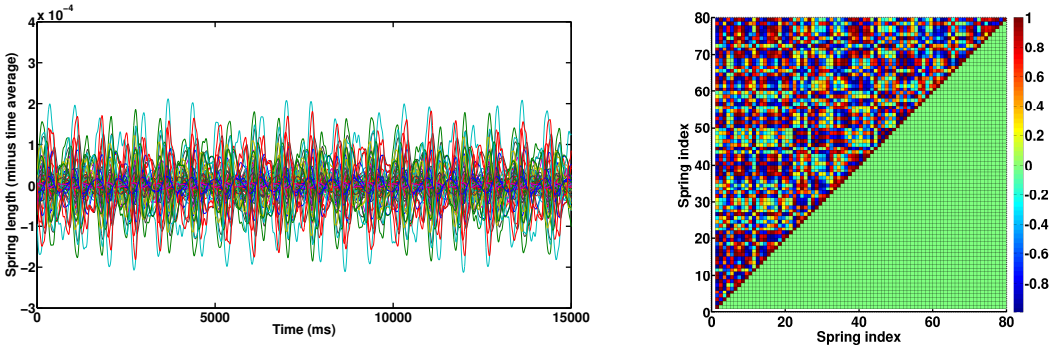


Figure 4.6 Spring lengths over time (left) and correlations between them (right) for a 30 node network with external inputs. For the time series of each spring length, we subtract from it its average value, so that we can see all the time series close to the y-axis. For correlations, we show the correlation of each time series with every other time series. In this case, both plots indicate a certain degree of diversity in spring states. The mean squared error for this network was 4.6067e-5.

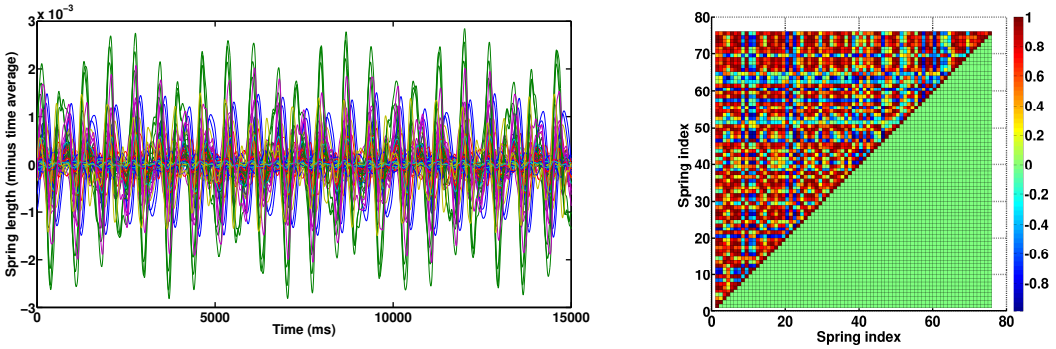


Figure 4.7 Spring lengths over time (left) and correlations between them (right) for a 30 node network with every spring used as an input actuator. In the correlation plot, we see what we might expect - a high degree of correlated activity across the network. This is also evident in the plot for spring lengths, where the magnitudes of oscillation vary, but their timings appear relatively synchronous, when compared to those in Figure 4.6. The mean squared error for this network was 1.7379e-4.

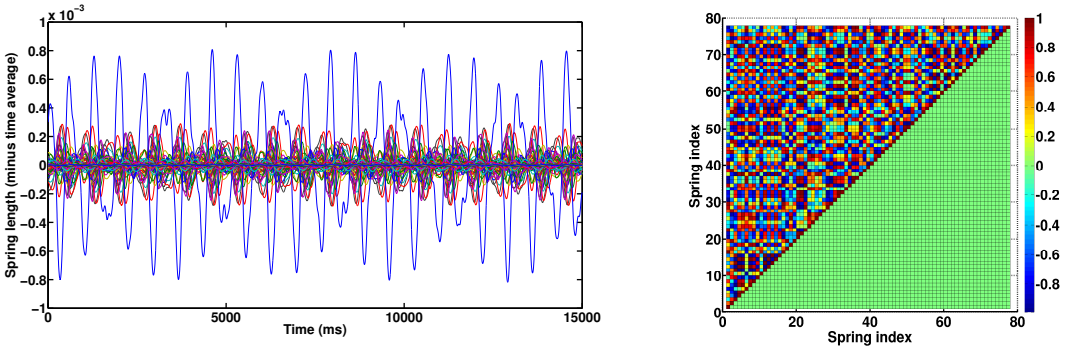


Figure 4.8 Spring lengths over time (left) and correlations between them (right) for a 30 node network with only one spring used as an input actuator. In this case, we see relatively high diversity in spring activity in both plots. The mean squared error for this network was 2.9699e-5.

transformation and temporal integration.

When we design a physical reservoir computing system to emulate some computational function, and simply map from input to output streams, we are at liberty to select the most effective parameters, including those for reservoir size and number of inputs to the reservoir. However, robotics and biology, and our virtual creatures, may be more constrained in their parameters. Therefore, we were interested to see what the effect would be of reducing the network size and of altering the proportion of network elements which were used as inputs to the system. We found that performance degraded gradually as the network size was reduced. Intuitively, we can explain this as a reduction in precision due to a reduction in the number of computing elements, although it is not straightforward to quantify such an effect in an analogue computing system. We were surprised to discover that performance is less dependent on the proportion of springs used as inputs to the network, even though this parameter has an effect on the degree of correlation in the activity of network elements. We believe this robustness is due to the nonlinearity of the network, and the variety in the dynamics of the MSDs.

These results show that, for a challenging task to emulate a Volterra series operator, we can actuate our networks internally, and we can also modify network size and the proportion of elements to be used as input actuators without losing all computational power. Therefore we have good reason to believe that the bodies of virtual creatures modelled with MSD networks, provided that they are large enough and diverse enough in their dynamics, will have similar power.

Conclusion

Here, we draw part one of the thesis to a close. This part of the thesis has been devoted to morphological computation with MSD networks. First, we showed that, as well as the analogue computation and central pattern generation which had already been demonstrated with these networks in a physical reservoir computing context, they can also be used to build controllers for animats and robots involved in behaviours which have been referred to as *minimally cognitive*. We showed that these controllers could be evolved to succeed in an active discrimination problem which can be solved with a purely reactive controller for an animat which drives left and right in a single dimension. We then proceeded to a similar task which was a step up in difficulty in several aspects: the controller had to control a robot in two dimensions instead of one, it had to cope with the addition of simulation noise and some environmental variation, and it had to deal with an explicit requirement for memory over a finite but extended period. We also introduced a new method for a persistent bistable memory element in MSD networks. These results together suggest that it is possible to construct, either 'by hand', via artificial evolution, or a combination of both, successful physical reservoir computers for the control of various purposeful and intelligent animat and robot behaviours.

In ongoing work, the beginning of which we describe in part two of the thesis, we continue to follow the thread of using and exploring the properties of MSD networks for the generation of behaviour. However, we now move to release them into the wild, not as controllers for mobile agents, but as the bodies of the agents themselves. As researchers including Schramm and Sendhoff (2011) and Joachimczak et al. (2016) have done before us, we are simulating swimming virtual creatures with internally actuated MSD network bodies.

In this chapter, we proved that internally actuated MSD networks perform comparably to externally actuated ones in emulating a Volterra series operator which requires nonlinear transformation and temporal integration. This implies that the bodies of our MSD network virtual creatures will have similar reservoir computing power, assuming they are large enough and have a variety of dynamics in their individual elements. Thus this chapter connects parts one and two of the thesis, with support for the hypothesis that our virtual creatures will potentially have all of the capabilities now known for MSD network physical reservoir computers.

Part 2 - Soft bodies swimmers and brain-body coevolution

Chapter 5

Soft-bodied Braitenberg Vehicle swimmers

A version of this chapter has been accepted for publication in Soft Robotics, under the title *Simulating soft-bodied swimmers with particle-based physics*.

While this chapter is largely focussed on behaviour and the introduction of our new subclass of morphological computation, which we name *morphological coordination*, the version which will be published is a slightly shorter piece, entirely focussed on our innovative methods for simulating swimming virtual creatures. We choose particle-based physics for this purpose, as it has the potential to allow a larger class of types of interaction between bodies and the surrounding fluid than the simple drag equations which tend to be used for this purpose. To minimise the computational costs of simulating particle-based fluids we have developed a method of only simulating fluids in a region local to the swimmers body; a kind of moving water tunnel which we call the *particle cloud*. We also introduce a technique for building soft bodies constructed as MSD networks with overlapping rigid body scales, a computationally cheap means of allowing for interactions with particle-based fluids, which we call *pseudo-soft*.

5.1 Introduction

Animal behaviour, in all its variety, is constituted by motions. In embodied cognitive science (Clark, 1996; Pfeifer and Scheier, 1999), the causes and coherence of those motions are explained as brain-body-environment interactions. When we focus on locomotion, we observe that, in general, animals move through space by means of the dynamical reconfiguration of their bodies. In legged locomotion, animals move their legs and their bodies follow; animals that fly at times glide with little effort, but also often flap their wings; animals that swim flex and deform their bodies in a multitude of ways to propel themselves through water. In other words, the morphology of an animal is not to be studied as a static pattern - its importance to behaviour may only be grasped by observing its motions and the dynamical properties of the body that give rise to them. To better understand the importance of morphology in generating purposeful behaviour, here we simulate the locomotion of soft-bodied swimming creatures to help shed light on how the body can be actively involved in the generation of adaptive sensorimotor behaviours. In particular, we examine how a continually reconfiguring body morphology, with very little in the way of explicit control, can act as a distributed locus of behaviour generation.

Previously, we have shown that, with the most minimal of artificial neural connections, simulated soft bodies can be employed as analogue computational reservoirs to generate appropriate control signals for simple mobile agents engaged in 'minimally cognitive' purposeful behaviour (Johnson et al., 2016). As biological soft bodies share some of the properties of our simulated ones, we believe this points to the possibility of similarly (analogue) computational cognition in nature. Here, however, we dwell less on notions of computation and set out to show that with

the correct mix of basic sensorimotor loops à la Braitenberg and the most minimal of neural and morphological coordination, a simple soft-bodied swimmer can achieve behaviour which exhibits a guiding purpose. We have chosen a 2-dimensional watery environment for our simulation, as for swimmers and fliers in fluidic environments, the entire profile of an agent's body, and its deformations, can be seen to be critical to determining its behaviour in a way which is sometimes less obvious for land-based behaviour. Our water is simulated with a novel particle-based fluid, which increases realism over models used in related research to date.

The branch of embodied cognitive science which is primarily concerned with these motions and their underlying dynamics is often referred to as 'morphological computation' (Paul, 2004; Pfeifer and Iida, 2005; Pfeifer and Bongard, 2006). It has already been found useful to speak of two classes under the general umbrella of morphological computation, one which is genuinely computational, as in (Hauser et al., 2011; Nakajima et al., 2013), and one which has been referred to as 'morphological control' (Füchslin et al., 2013), to cover cases which deal more with control, for example of the body of an animal or a robot, than with computation in the information processing sense. There are also cases, such as the swimmers we report on here, which would seem to fall under the morphological computation umbrella but are not easily described as examples of computation nor control, and for these we introduce a third class, 'morphological coordination'. In fact, none of these classes should be considered mutually exclusive, but, as will be discussed further in **Discussion**, the notion of morphological coordination applies well to cases such as ours, where the controller is minimal, and cases such as the passive dynamic walker (McGeer, 1990), which are entirely absent of control systems.

This chapter is structured as follows: we will begin with a brief review of related work in **Related work**, then we will describe simulation methods and the design and controllers used for our swimmer in **Methods**, followed by the results of running the simulation in **Results**, and some analysis of how the swimmer is physically able to turn and to propel itself through the water in **Motion analysis**. Finally, we will close the chapter with a discussion of the importance of purpose to locomotion, how morphological coordination can play a part in determining and achieving purpose, and how our swimmer fits into this picture in **Discussion**. We posit that with this platform and the theoretical underpinnings of morphological computation and morphological coordination, we can deepen our understanding of the role of the body in generating purposeful and intelligent behaviours and the importance of morphology to animal behaviour.

5.2 Related work

In Braitenberg's book, *Vehicles* (Braitenberg, 1986), the behaviours of his simplest vehicles arise in a synthetic model, built from the bottom up, of highly abstracted sensorimotor loops and sources of stimulation. In the style of the cyberneticians, Braitenberg refers to his own creations as machines, but their conceptual descendants are often categorised as artificial animals, or *animats* (Wilson, 1991). Our own animat has dual descent from Braitenberg's vehicles and Sims' *virtual creatures* (Sims, 1994), which were evolved in a physics based simulation, by way of numerous in silico studies of embodiment and the importance of morphology, including those in (Bongard, 2010; Rieffel et al., 2014; Auerbach and Bongard, 2014; Shim and Husbands, 2012, 2015). In our experiment, we simulate an animat in a watery environment with a two-dimensional physics engine which includes both rigid bodies and particle-based fluids. The basic structure of our animat is triangular, because it is inherently stable and simple to work with. The animat swims towards or away from the source of a chemical gradient, depending on the nature of the connections from sensors to actuators, by continuously modifying the lengths of its sides in a coordinated fashion.

Since Sims' virtual creatures, there have been many experiments conducted with rigid-body physics engines. Due to the additional technical challenges involved, experiments using soft-body physics have somewhat lagged behind, although significant progress has been made recently. We will begin our short review with examples of three-dimensional simulations which have used

freely available physics engine libraries such as *ODE* (Smith, 2016), *Bullet* (Coumans, 2016) and NVIDIA's *PhysX* (NVIDIA, 2016). Authors including Rieffel et al. (2010); Mirletz et al. (2014); Caluwaerts et al. (2013) and Hustig-Schultz et al. (2016) have simulated locomotion with compliant actuated tensegrity bodies, using *ODE* in the first case, and the *NASA Tensegrity Robotics Toolkit (NTRT)*, which is based on *Bullet* in the remaining three. Wittmeier et al. (2011) developed *CALIPER*, a simulation framework for tendon-based robots, and Diamond (2013) has simulated the entire upper-body anthropomorphic musculo-tendon robot *EcceRobot*, in both cases by combining rigid body dynamics from *Bullet* with customisations to allow the integration of elastic cords. Moore et al. (2015) have used *ODE* to simulate an animat with a flexible spine, implemented with spring-like joints. Rieffel et al. (2014) have shown how to evolve and actuate soft bodies in *PhysX* with tetrahedral primitive elements. Lessin and Risi (2015) have also used the *PhysX* soft body physics to implement muscular organs attached to rigid body skeletons. Doursat and Sánchez (2014) have used *ODE* in the simulation of soft-bodied robots with a cellular basis. There are also cases where custom models have been used rather than physics engines, as in the octopus arm experiment of Nakajima et al. (2013), and the early examples of Albert's tubular animat (Albert, 1999) and the artificial fishes of Terzopoulos et al. (1994). Also, Hiller and Lipson (2014) have recently introduced a new physics engine, *Voxelyze*, which enables the creation and actuation of soft bodies made of voxels with arbitrarily varying dynamic properties, which has been used for land and water-based locomotion studies, respectively, by Cheney et al. (2014) and Corucci et al. (2016). In related experiments which have been conducted in only two-dimensions, it has been the norm to avoid existing physics engines and implement more compact and computationally cheap models, as in Sfakiotakis and Tsakiris (2006) and Schramm and Sendhoff (2011). Notable exceptions to this rule include work by Joachimczak and Wróbel (2012), which incorporated *Bullet* rigid body physics into a simulation of soft swimmers, and the *Soft Cell Simulator*, from Germann et al. (2013), in which deformable membranes are simulated with flexibly linked chains of small rigid bodies using Catto's *Box2D* (Catto, 2013) physics engine.

In contrast to the above work, we focus on the role of body morphology dynamics in the generation of behaviour. To that end, we present a soft-bodied animat engaged in chemotactic behaviour in a virtual watery world. Earlier swimming virtual creatures have tended to be of low to medium morphological complexity, in terms of the number and interconnections of the parts of the body. Here, we go to around the lowest complexity possible to illuminate the minimum requirements for our animats to succeed in their tasks. As simplified and abstracted from the real world as this simulation is, we believe that these requirements may be similarly simplified and abstracted from those for the directed locomotion of animals and robots.

Our simulation is technically novel in two ways. First, this is the first virtual creatures experiment to make use of particle-based physics to simulate water in a more detailed and realistic fashion than is the norm. Second, similar to Germann et al. (2013), we have developed a way to simulate a soft-bodied animat using only rigid bodies, through the use of overlapping scales.

5.3 Methods

5.3.1 Platform

Google's *liquidfun* (Google, 2013) a two-dimensional physics engine library in the C++ language, which extends *Box2D*, was used as the basis of this simulation. *liquidfun* adds soft bodies and fluids, both implemented with particle-based physics, to the rigid bodies already present in *Box2D*. We selected a particle-based fluid model as it encompasses a wider range of dynamics than simpler models such as that of (Sfakiotakis and Tsakiris, 2006). In particular, as well as causing drag on objects moving through it, a particle-based fluid can also exhibit its own dynamics, such as waves and turbulence.

A simulation interval of 1 ms was used, for stability and realistic rigid body and fluid inter-

actions. In this interval, a single iteration is specified for each of velocity, position and particle iterations. The particles used in our simulation are all of the water particle type, with radius 0.125 m and a density of 0.005 kg/m^2. These rather peculiar parameter values were arrived at for the following reasons. Although its units are essentially arbitrary, dimension in *Box2D* are labelled as SI units and we have retained that convention. *Box2D*'s physics for rigid bodies and the various joints which may connect them can become unstable at very small scales, and so to avoid this problem, our swimmer's body, described in more detail below, has a side length of 2 m. The scale of the swimmer being set, the particle dimensions were obtained by trial and error, with the competing objectives of realistic fluid-body interactions (assessed only by visual inspection so far) and computational performance in mind. In particular, if particle radii are too large, we cannot expect realism, as the fluid will be made of large balls which only make contact with the swimmer's body at a small number of points, but if, on the other hand, they are too small, their number will be much larger, which will impact heavily on performance.

5.3.2 Morphology

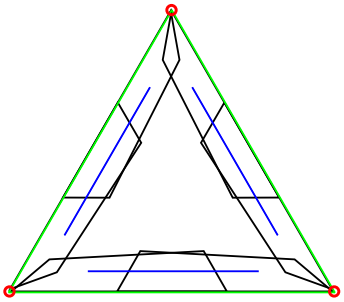


Figure 5.1 The swimmer's body is shaped as an equilateral triangle, enclosed by three pairs of overlapping scales (black lines). Small circular nodes are placed at the vertices of the triangles, for the attachment of the scales and passive and actuated distance joints. The scales are joined by revolute joints at the vertex nodes (red circles), and by prismatic joints (blue lines). Damped springs (green lines), implemented as distance joints, also connect the vertices. The lengths of the left and right faces are controlled by actively modifying the rest lengths of their respective springs, while the back face is passively compliant.

In order to avoid confusion, when we refer to 'vehicles' we are referring only to those of Braitenberg, and specific vehicle designs will be referred to with the following form: Vehicle 1b. From here on, we will refer to our own animats as 'swimmers', and for specific variants will adopt the following form: Swimmer A.

Our swimmer variants, described below, differ from one another only in terms of the connections between sensors and motors, and are identical in all other respects. As seen in Figure 5.1, a swimmer's body shape is an equilateral triangle with side length 2 m. In the current configuration, the lengths of the active sides contract and expand equally about this rest length. As has been previously shown to be effective in related work including (Terzopoulos et al., 1994; Rieffel et al., 2010; Caluwaerts et al., 2013; Joachimczak et al., 2016), the active joints are energised by smoothly changing the rest lengths of their springs. While *liquidfun* implements soft bodies, it does not provide any straightforward way to attach them to other bodies or to dynamically control properties such as shape, or stiffness and tension. Therefore we developed a novel way of emulating a soft body through the use of overlapping scales. Each of the body's faces has two overlapping polygonal rigid bodies, or scales, which can slide past one another without collision. This means that, within certain bounds, the dimensions and area of the triangular body can change with the faces being kept flat and in constant contact with the surrounding particles. Inside the scales, the swimmer's body is hollow.

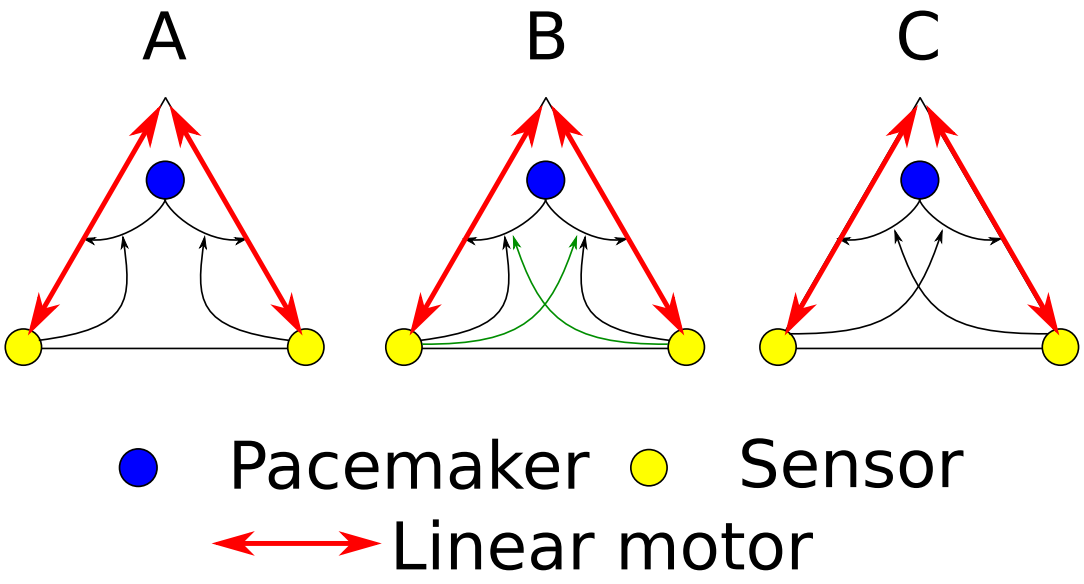


Figure 5.2 Swimmers A to C, shown from left to right. A central pacemaker, drawn as a blue filled circle, synchronises the motion of the left and right actuated faces. The phases of oscillation are modulated by the output of the sensors, drawn as yellow filled circles, placed on the vertices of the back face. Direct connections lead to positive taxis, as in Swimmer A and Swimmer B, and crossed connections lead to negative taxis, as in the case of swimmer C. Swimmer B has additional crossed connections from the sensors which control the amplitudes of oscillation. The connections which modulate phase are shown in black, and the connections which modulate amplitude are shown in green.

In a preliminary implementation, it was found that if the swimmer's body cavity was empty, the passive distance joints would needed to be stiffened to prevent the body from collapsing. In other words, due to the energy in the particle system which comes from a repulsive force between particles, it was necessary to equalise the pressure between the interior and exterior of the swimmer by allowing particles to fill its cavity. Although the current parameters for the swimmer's body make it withstand water pressure even if its cavity is empty, allowing the body to fill with particles as they are added to the simulation has no noticeable effect on its behaviour, and so in the results presented here that has been allowed. However, in our simulation, individual particles occasionally have very high energy, which can lead to a particle tunnelling through the swimmer's scales, in either direction. With the present configuration, tunnelling particles typically escape rather than enter the body, as they are compressed by its contraction or caught between the swimmer's scales at the inside corners of the body and then forced out. The high energy caused by these events is in evidence in Figure 5.10, where some particles inside the swimmer's body have high velocities.

Circular nodes with diameter 0.01 m and mass 1 kg are placed at the vertices of the triangle, for the attachment of the scales and various joints. The nodes are attached to each other along the three sides of the triangle with distance joints, two of which are actuated, which have the dynamics of damped springs. Scales are attached to the nodes with revolute joints, which have no constraints or dynamics. Scale pairs on each side are attached to each other with frictionless prismatic joints, to keep the faces flat as the scales slide past one another. Each scale has a mass of approximately 0.25 kg. The springs on the active edges of the swimmer's body have natural frequency of 100 Hz, while the spring in the passive edge on the back of the swimmer has natural frequency 10 Hz. All springs have a damping ratio of 1.

The swimmer's sensors are located on the nodes on its back edge. These abstracted sensors directly apprehend the distance from the nodes to the source of the gradient.

5.3.3 Control

The swimmer swims by actuating the distance joints on the left and right sides of its body. As in the jellyfish (Katsuki and Greenspan, 2013), and the evolved creatures of Cheney et al. (2014), internal coordination of the swimmer's motion is achieved by the use of a pacemaker. The active joints are energised by continuously changing the rest lengths of their springs according to a sinusoidal function of time, driven by the pacemaker. For all swimmers, the frequency of oscillation is approximately 14.32 Hz. We found that with the particular parameters of our system, it was not necessary to apply feedback control to the joint lengths, as the actual lengths of the joints tracked the specified rest lengths with only a small lag. The pacemaker is a sinusoidal signal of fixed frequency which is sent from a notional central circuit to both actuators. The amplitude and phase of the signal may both be modulated at each actuator according to the sensor outputs. We implemented three different control arrangements, detailed below and referred to as Swimmers A, B and C, and shown in Figure 5.2.

Swimmer A has a constant amplitude of oscillation, of magnitude 0.5 m. For this swimmer, the connections from sensors to actuators are uncrossed and the phases of oscillation are equal to the respective sensor outputs. When the phases of oscillation between the two sides diverge, the swimmer tends to turn, as detailed in **Turning**. When the sensors are equally distanced from the gradient source the oscillations are synchronised and the swimmer will travel in a straight line. Theoretically, in a noise-free model, a Braitenberg vehicle which is wired for positive taxis may drive down rather than up the gradient of a stimulus, if it is oriented directly away from the stimulus source, due to the fact that such a vehicle only turns when there is a difference between its two sensor activations. We found that this is unlikely to be the case (and is as yet unseen) if we start our simulation with Swimmer A oriented directly away from the source of the chemical gradient. What we tend to see is the swimmer turn slightly, presumably initially nudged by the small motions of the surrounding particles, and then proceed to swim backwards, looping past the sensor stimulus. This peculiar behaviour, an attractor in the agent-environment dynamics which does not appear easily escaped, does not appear in the results presented here, where the swimmers' initial positions and orientations were selected randomly. The typical behaviour occurs at both long and short ranges from the source, and so this swimmer never stops swimming, but, as can be seen in Figure 5.6, will tend to make small excursions around the source of the gradient, once within its vicinity.

Swimmer B has the same steering mechanism as Swimmer A, but has an additional function to control the amplitude of oscillation, so that when it is close to the gradient source it will expend little or no energy in continuing to swim. This function is a nonlinearity which has little effect beyond short distances from the gradient source, but will stop the swimmer swimming when it has reached the source. In the current implementation, when the swimmer reaches a point close to the gradient source it will tend to stay very close to it, with its body rotating in position, under momentum for part of the time and actuation for the rest, as one or both of its sensors passes in and out of the region in which the stimulus amplitude falls to zero.

As with Braitenberg's vehicles, the behaviour of a swimmer can be switched from positive to negative chemotaxis by merely switching the sensor connections from straight to crossed. Swimmer C is the same as Swimmer A except that the connections which control the phase of actuation are now crossed, which causes it to swim away from the the source of the gradient.

Although the general behaviour is determined by the nature, crossed or uncrossed, of the connections between sensors and actuators controlling the phase of oscillation, the same is not true for the functions controlling amplitude. As long as the amplitude of oscillation decreases as the distance from the gradient source decreases, the behaviour of a swimmer is qualitatively unchanged by crossing or uncrossing the relevant connections.

For all swimmers, the underlying actuator functions are:

$$restlength_L = baselength + A_L + sin(90t + \phi_L) \tag{5.1}$$
$$restlength_R = baselength + A_R + sin(90t + \phi_R) \tag{5.2}$$

For Swimmers A and Swimmer C, the amplitude of oscillation is a constant value:

$$A_L = A_R = 0.5 \tag{5.3}$$

.

whereas for Swimmer B the amplitudes of oscillation are nonlinear functions of the sensor outputs:

$$A_L = \begin{cases} 0.5 - \frac{1}{sensor_R^2}, & \text{if } sensor_R \geq \sqrt{2} \\ 0, & \text{otherwise} \end{cases} \tag{5.4}$$

$$A_R = \begin{cases} 0.5 - \frac{1}{sensor_L^2}, & \text{if } sensor_L \geq \sqrt{2} \\ 0, & \text{otherwise} \end{cases} \tag{5.5}$$

For Swimmer A and Swimmer B, the phases of oscillation are equal to the sensor outputs:

$$\phi_L = sensor_L \tag{5.6}$$
$$\phi_R = sensor_R \tag{5.7}$$

whereas for Swimmer C, the same is true but the connections are crossed:

$$\phi_L = sensor_R \tag{5.8}$$
$$\phi_R = sensor_L \tag{5.9}$$

5.3.4 Simulation time optimisation

In our simulation, the computational cost of simulating particle-based fluids is high, and can slow execution considerably. Although this swimmer was hand-designed, as its parameters are few enough to be amenable to hand-tuning, the methods developed here are intended to be extended and used in artificial evolution, which requires numerous simulations to evaluate behaviours. For this reason, it is essential to minimise the computational cost of simulating very large numbers of particles. An ad-hoc solution to this problem, is being developed, the first iteration of which is presented here.

In *liquidfun*, particles are grouped into systems. For a given particle density, the number of particles to be simulated is directly proportional to the area they fill. The computational cost of a particle system cannot be precisely predicted, as, for a given number of particles, the computational cost of simulation is variable, depending on how much energy there is in the system. However, as area is related to the square of distance, so the relationship between the scale of a particle system and its computational cost is a quadratic one. This being the case, significant performance gains can be made by minimising the scale of a particle system.

We begin with the assumption that for any particle or body there will be some distance beyond which the surrounding particles have only a negligible effect (see Figure 5.10 for an illustration).

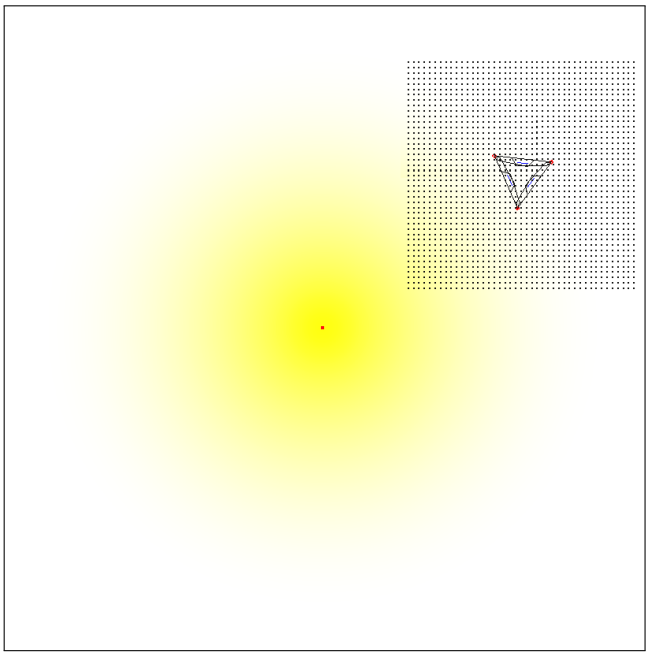


Figure 5.3 The reduced particle system, seen in a screen capture from a simulation debugging window. To enable quick development with a relatively fast simulation, only a sub-region of the simulated world, centred on the swimmer's body, contains particles. The particles are free to move within the square-shaped region, but the boundary moves with the swimmer.

As the swimmer detects only the local concentration of the chemical gradient, this region is an approximation to the swimmer's *umwelt* (Von Uexküll, 1957), as it contains all which the swimmer can detect, affect, and be affected by directly. Based on this assumption, the first step in our ad-hoc minimisation was to define a system of particles in only a sub-region of the tank, centred about the swimmer, and moving with it. We refer to this system, loosely based on the idea of a moving water tunnel, as the 'particle cloud'. Notionally, this system is square-shaped, and the particles are prevented from dissipating and escaping the system by exerting an inwards force on any particles which travel outwards beyond the boundary. In practice, the forces are kept as small as possible, so as not to cause large turbulent effects, and so only weakly constrain the shape of the particle system. An issue which arose during the development of this method is that a positive feedback loop can form between the motion of the swimmer and that of the particle system. As the boundary changes position with the swimmer, it can also effectively cause a current in the direction of the swimmer's travel, increasing the swimmer's motion, and so on. This is particularly noticeable when the swimmer is passively drifting.

In order to counteract this effect, a current is artificially induced approximately in opposition to the swimmer's motion. At the beginning of every simulation step, a randomly selected particle is deleted, and a replacement created outside the particle system, in front of one of the edges which the swimmer is moving towards. As the replacement particles are driven into the system by the aforementioned confining force, the particle system is internally agitated. When the swimmer is moving in a relatively straight and quick fashion, an opposing current in the particle system can be observed. As the rate of input of new particles to the system is constant, rather than based on the speed the system moves at, when the swimmer is moving slowly or with less clear direction, a more turbulent effect can be seen. Overall, although there are some differences between the dynamics of the ad-hoc simulation and one where a tank-full of particles are simulated, they are similar enough that it appears that most, if not all, combinations of swimmer parameters and control structures which prove effective in the ad-hoc environment also prove effective in the other.

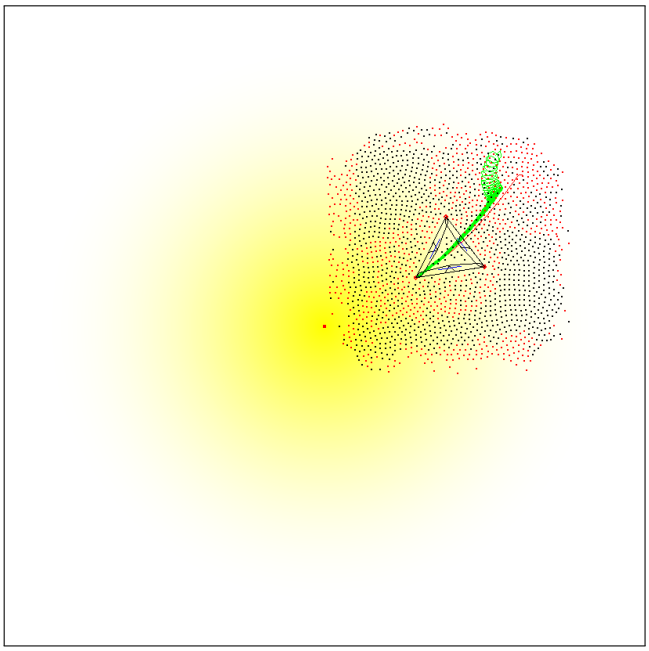


Figure 5.4 As the swimmer moves, the particle system is loosely constrained to track its position. Newly created particles and particles which are found outside the boundary due to either their own movement or that of the swimmer have a force applied to restore them to the defined area. The forces are kept as small as possible, so as not to cause large turbulent effects, and so only weakly constrain the shape of the particle system. The colour of new particles periodically alternates between red and black so that currents can be observed. Although it is not easily seen in a single frame, when the swimmer is swimming in a reasonably straight line a current opposing its motion can often be observed. The red and green traces behind the swimmer indicate the positions of its front node and centre of mass, respectively, over time. Videos of the swimmers are available for viewing at https://www.youtube.com/channel/UCkJR61yyoIgOzFWWIawHUPg

Indeed, although it is possible that there will be effects present in the particle cloud but not in the full simulation which an swimmer's performance depends upon, in general the artificially induced current appears to make the particle cloud the more challenging of the two environments, meaning that any swimmer that succeeds in its task in the particle cloud should also succeed in the larger particle system.

One final problem arose from the fact that the plane of operation of our swimmer is perpendicular to the axis of gravity, which is to say that the simulated particles are unaffected by gravity. This results in the particle system having low pressure and as a result being highly compressible. In these circumstances, as the swimmer expands, it pushes the particles away from it, but they return more slowly than its following contraction, meaning that not all faces of the swimmer are in contact with the water at all times. In order to counteract this effect, and artificially increase water pressure, the particle cloud is constrained in an area which is approximately half of the area it is instantiated in. In the simulation with a full tank, this is achieved more directly by simply creating two particle groups in the same space.

5.4 Results

In this section, we begin by evaluating the gains in performance made through the use of the particle cloud, and then take a look at the behaviours achieved by our swimmer's three different Braitenberg-style controller variants.

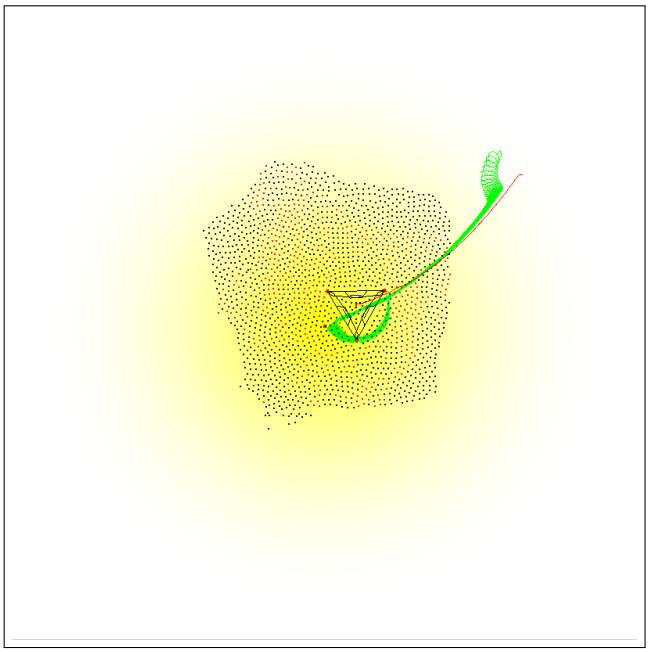


Figure 5.5 When swimmer B gets close to the source of the gradient, it ceases to change position, but tends to rotate on the spot. At this point, the flows in the particle system become more turbulent, most clearly evident from the way the boundary becomes more irregular.

5.4.1 Performance

Simulations were run on a linux machine with an AMD FX-6300 six core CPU and 16 GB of RAM. *liquidfun* is not parallelised, and so the simulation runs on a single processor core. When collecting data for this chapter, a significant amount of data, amounting to almost 17 MB for each run, was saved to file, but this process ran on a second core and only increased the cost of simulation by around 10 %. When the particle cloud method is employed and there is no animation displayed, 90 s of simulated time can be executed in approximately 40 s. Memory usage is negligible, at approximately 30 MB. When a square tank of particles is simulated, with side length of 20 m, the time taken for a run of the same simulated duration took approximately 11 minutes. Therefore, in this case, the use of the particle cloud increases efficiency by a factor of 16.5.

It should be noted that when using the particle cloud method, for a given particle density, the cost of simulation is related only to the size of the swimmer. If the swimmer is scaled up, the number of particles required to envelop it will also rise, quadratically, as noted in **Simulation time optimisation**. On the other hand, the spatial extents of the swimmer's environment have no effect on the size of the particle cloud, and so can be increased arbitrarily. In fact, where the particle cloud was used the swimmers were initialised within the same same region enclosed in the tank scenario, but the walls were not included in the simulation, and so in some cases the Swimmer C, which is averse to the gradient source, travelled far beyond the tank's notional extents (Figure 5.8).

5.4.2 Swimmer Behaviour

In all of the following, swimmers begin at a randomly generated orientation and location inside a square-shaped region centred on the source of the gradient, with side length 16 m. The simulation is then run for 90 simulated seconds, but the swimmer is not actuated for the first 700 milliseconds, in order to give the particle system time to stabilise. This stabilisation period is required because

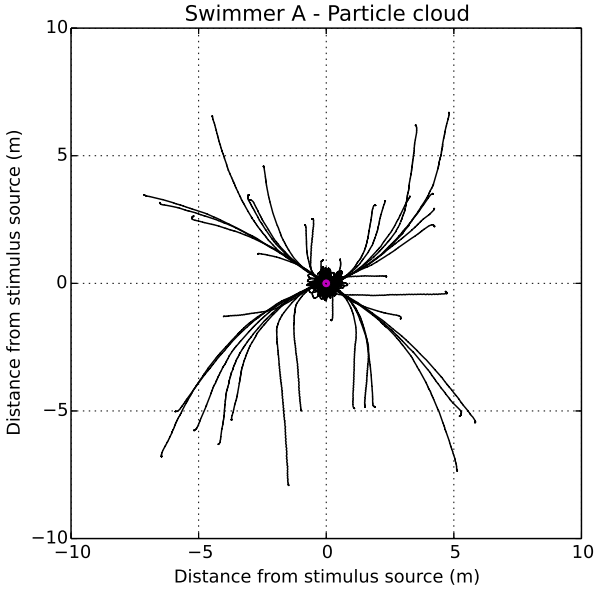


Figure 5.6 Trajectories of Swimmer A. Swimmer A, similar to Braitenberg's vehicle 2b, swims towards the source of the gradient, but does not stop when it reaches it. However, given how sensitive it is to a difference between sensor outputs, nor does it travel far beyond the source. Instead, it tends to keep circling back. In this figure, and the following three, the source of the chemical gradient is at the origin, and the swimmer's central position over time is shown in the traces. The swimmer is started from random positions and orientations. Clear trends in the effect of water current on the motion of the swimmer can be seen, with many trajectories pulled towards paths parallel to the x and y axes, and the lines which pass through zero and divide the quadrants of the x- and y-axes.

the methods of increasing pressure by doubling the number of particles introduce high energy to the particle system, which is not initially balanced by even distribution. The source of the chemical gradient is not modelled as a physical body, and so the swimmers cannot crash into it. All swimmers were simulated for 40 runs with the particle cloud. Swimmer B was also simulated for the same number of runs in the water tank, for comparison. So that the three controllers and the two simulator variants can be compared directly, we used the same set of initial positions and orientations for all sets of runs.

Swimmer A was designed to behave like Braitenberg's Vehicle 2b, which he also referred to as AGGRESSOR. As can be seen in Figure 5.6, the swimmer always finds its way to the source of the gradient. However, while Vehicle 2b will theoretically sail straight by the stimulus source, the swimmer does not. When the swimmer is swimming towards but distant from the source, if the dynamics of the water push it slightly off course, the difference in distances from the two sensors to the source will be relatively small. The resulting phase difference between the oscillations of the two joints will lead to small corrective motions. However, when the swimmer is close to the source, small angular perturbations from the correct course will lead to more pronounced differences between the two distance measurements, and the swimmer will begin to turn. This causes the swimmer to get trapped in the source's vicinity, often missing the source by a short distance but always turning back towards it.

Swimmer B is modified from Swimmer A to behave more like Braitenberg's Vehicle 3a, which he referred to as LOVER. It has the same mechanism as Swimmer A, and so steers and swims towards the chemical source in the same way, but as described in **Control**, it has additional sensorimotor loops added to inhibit actuation when the swimmer is close to the source. Like Swimmer A, this swimmer does not entirely halt its swimming action when it reaches the source, although

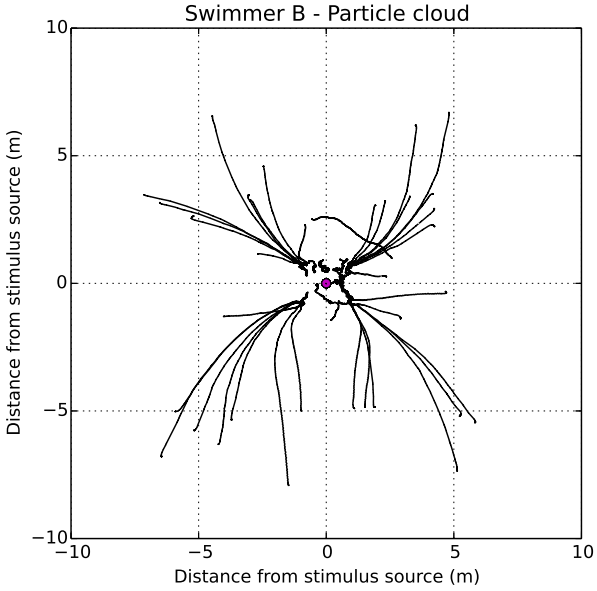


Figure 5.7 Trajectories of Swimmer B. The behaviour of this swimmer is very similar to that of Swimmer A, except at short distances from the source of the gradient, where Swimmer B will slow down and eventually stop. It can be seen that the controller is not infallible, with one run, where the swimmer started at coordinates close to [-1, 2.5], resulting in the swimmer circling the gradient source rather than approaching it.

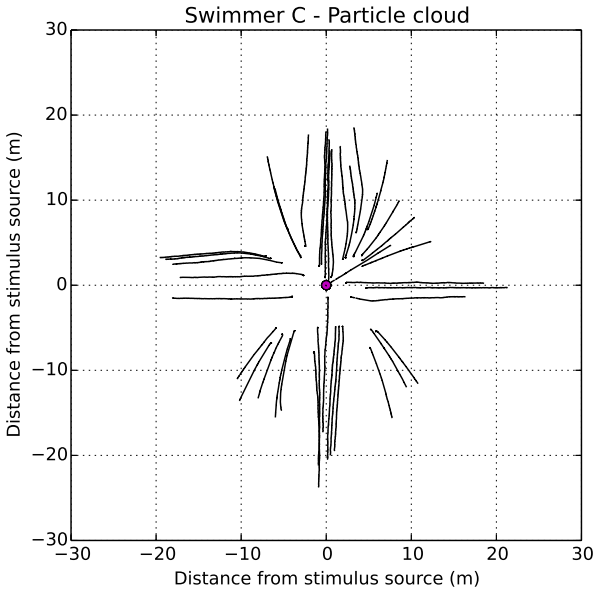


Figure 5.8 Trajectories of Swimmer C. Swimmer C is an avoider. As for Swimmers A and B, currents in the particle cloud often have a pronounced effect on the swimmer's trajectories.

it does tend to expend less energy.

As Swimmer C is modified from Swimmer A by crossing the sensorimotor connections which determine the phases of oscillation, it will avoid and swim away from the chemical source, like Braitenberg's Vehicle 2a, the COWARD.

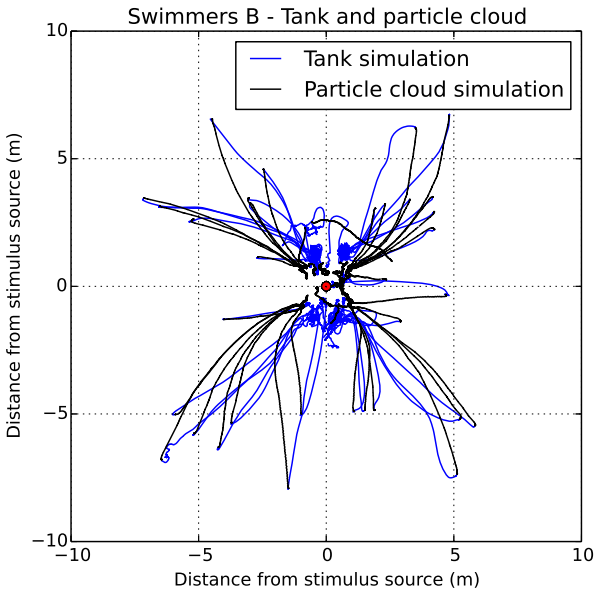


Figure 5.9 Trajectories of Swimmer B in both simulation variants. Apart from the one failed run in the particle cloud simulation, the behavioural differences are minor, with the swimmer's steering clearly affected by the different environments, but not enough to change the overall behaviour.

In Figure 5.9 it can be seen that the behaviour of Swimmer B in the two scenarios (particle cloud simulation and full tank simulation) is qualitatively similar, and successful for almost all initial conditions. However, it is also apparent from Figures 5.6, 5.7, and 5.8 that the current method of injecting particles to the cloud is causing some bias in the swimmer's trajectory, and this is particularly evident for Swimmer B (Figure 5.7), where the water currents tend to flow towards the source, at angles which divide the quadrants formed by the x- and y-axes.

5.5 Motion analysis

In this section we perform some simple analysis of how our swimmers are actually able to swim and to turn. Our objective here is not a complex or detailed analysis of the fluid dynamics involved in this, or of the swimming gaits, but rather to form a general overview of how this streamlined swimmer actually locomotes. We do this by breaking the sensorimotor loop, and controlling the body of a swimmer first to simply swim forwards, and then to rotate its body on the spot. In other words, we focus on the potential of the body here, not the swimmers with different controllers that we have examined above.

5.5.1 Forward motion

To gain some insight into how the swimmer moves, we first examined the case of swimming straight forwards. In this case, the two active sides of the body expand and contract synchronously. When the swimmer is activated with no particles in the simulation, it tends to slowly drift about its centre, but does not change its position. When we restored the particles to the simulation, and lowered the frequency of the actuation cycle from 14.32 Hz to only 1 Hz, we found the same thing. From the observation that the swimmer only moves forwards in the presence of particles, we can conclude that in the energy exchange between swimmer and fluid, the successful swimmer wins, and that some of the energy transferred from it to the surrounding fluid is returned to push it

forwards. In Figure 5.10, which shows the trajectories of particles and their velocities for a single actuation cycle of a swimmer at both tested frequencies, we see that at the lower frequency, the energy of particles around the swimmer's body is concentrated in much the same way relative to each of the swimmer's outer faces. This explains why the swimmer does not move in this case. In the case of the higher frequency, it is clear that energy is more concentrated behind the back face than it is to the sides, which explains how the swimmer is pushed forwards. While we do not consider the physics engine we use here to be completely realistic in all aspects, these contrasting effects appear to be broadly consistent with one aspect of swimming in biology, where, as noted by Purcell (1977), reciprocal swimming motion like that of our swimmer is only effective at higher Reynolds numbers.

As well as this forwards motion that we designed for, we have discovered that the swimmer can also swim directly backwards, in the case that its two actuated sides oscillate exactly in anti-phase.

5.5.2 Turning

While, for our swimmers, travelling has a clear dependence on the surrounding fluid, we discovered that turning does not. We examined this case by setting the phase difference between the two oscillators to 90 degrees. When the right oscillator leads, the swimmer turns left, and vice versa. We set the swimmer to turn in three separate conditions: in the tank, in the particle cloud, and with no particles. In all cases, the motion was qualitatively the same, although the turning rate varied. The swimmer's turn is fastest in the tank condition, followed by the particle cloud, and lastly the no particles condition. From this we conclude that although, as in traveling, the turning swimmer receives an energy rebate from its interaction with the surrounding fluid, which can increase its turning rate, the general turning motion results from the motion of the swimmer's own constituent bodies, when they are moved in the appropriate relative phases.

Figure 5.11 shows the turning motion of the swimmer for a single oscillation cycle during a left turn. The duration of the turn is split into four quarters, and for each quarter the initial profile of the swimmer is shown in blue, and the final profile in red. Quarters 2 and 4 are periods where both active sides contract and expand, respectively, and do not alter the swimmer's position or orientation considerably. Therefore it is apparent that the swimmer's turn over a single cycle is the net effect of quarters 1 and 3. As the two active sides oscillate with a 90° phase difference to each other, the swimmer turns first one way and then the other, but the second turn is slightly larger than the first. This is because in quarter 3 the side lengths are shorter than in quarter 4, and so the motion of the nodes, which has the same magnitude in both cases, leads to a more pronounced effect in the second. In both cases (forward and turning motion) it is clear that it is the dynamical interactions between environment and the continually changing body morphology that are at the heart of the swimmer's motion.

5.6 Discussion

In this concluding section, we will begin by stressing the importance of purpose to locomotion. Next, we will go on to consider how not only the nervous systems often assumed to control bodies, but also the properties of bodies themselves, can coordinate locomotive behaviour. Those points being made, we will then return to our swimmer, with insights into its behaviour of a more general and theoretical nature than the preceding analysis. Finally, we will summarise key details of our simulation, and its motivation.

When an animal moves through space, it generally does so to either approach or move away from objects and other animals. Even during a period where an animal has a single resolute purpose, it is possible that surrounding objects and animals are simultaneously moving around it. Therefore, not only is locomotion purposeful behaviour, but if it is to be successful it requires the ability to not only travel in a straight line, as if everything desirable was in front of the subject

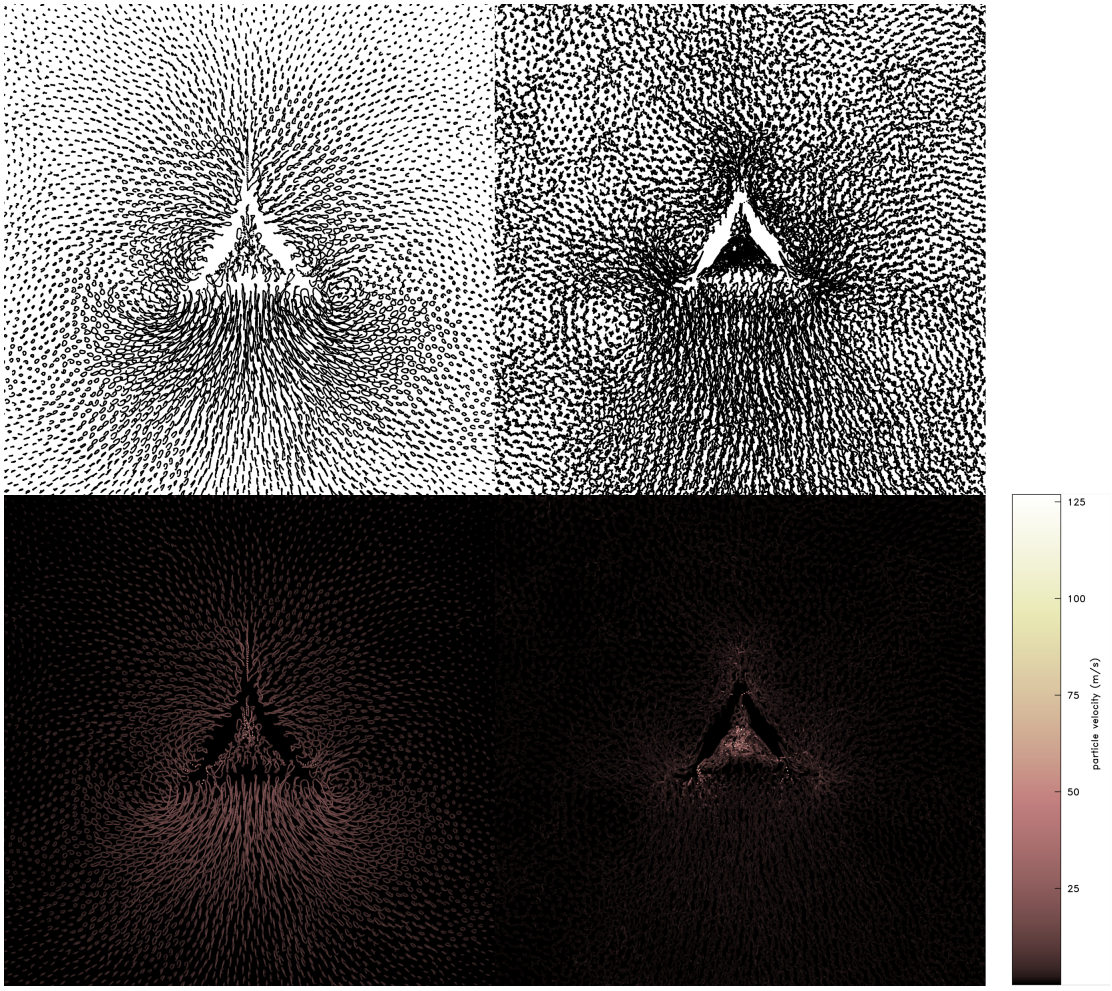


Figure 5.10 In this figure, the trajectories of particles are shown. In the top row, the traces show the positions of particles over time. In the bottom row, the same traces are shown, but colour is used to illustrate the velocities of the particles. The left column shows a single cycle of oscillation of the swimmer's active joints, when the pacemaker is activating the joints at the normal frequency of 14.32 Hz. In the right-hand column, the same is shown for a swimmer with a pacemaker driving oscillation at the reduced frequency of 1 Hz. When we compare normal swimming to low frequency swimming, three points of contrast between the two stand out: First, the higher frequency oscillation imparts significantly more energy to surrounding particles than the lower frequency one, per oscillation cycle. Second, this higher energy leads to more structure in the motion of the particles in the higher frequency case. Third, the higher frequency case shows more energy behind the swimmer, on average, than on its sides, but the lower frequency case shows fairly uniform concentrations of energy on all sides of the swimmer. This last observation is consistent with the fact that the swimmer moves forwards with the 14.32 Hz pacemaker, but does not with the 1 Hz one. A final point to be made here is that in both cases, the effect of the swimmer's motion on the surrounding particles, and vice versa, falls to almost nil at approximately 1.5 body lengths from its exterior faces. This led to the realisation that it may be possible to minimise the number of particles in our simulation, as described in **Simulation time optimisation.**

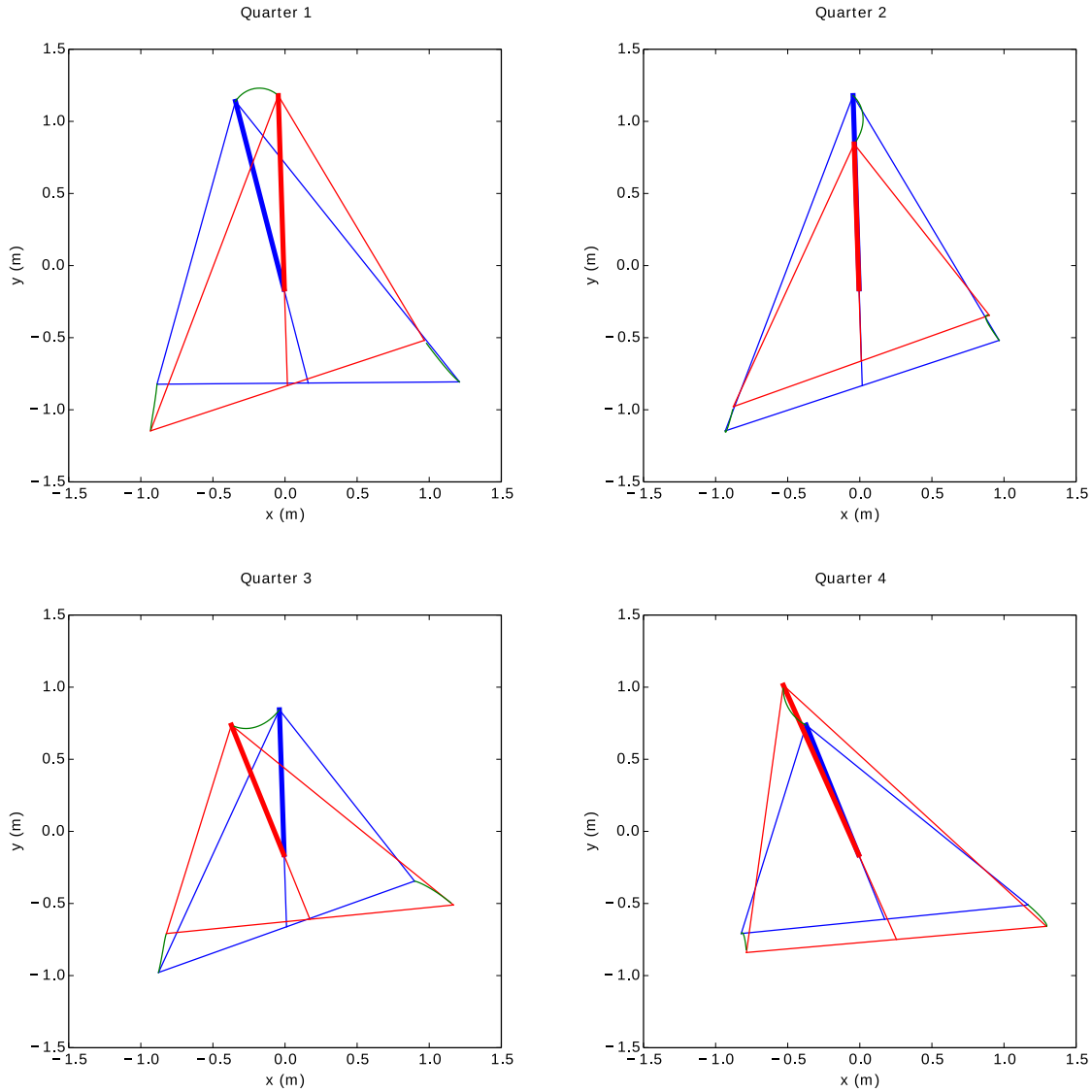


Figure 5.11 Turning. The four figures here show the motion of the swimmer over a single cycle of oscillation, split into four quarters. In each figure, the blue lines show the swimmer's profile at the beginning of the quarter, and the red lines show the profile at the end. The green lines connecting respective apices show the trajectories of the apex nodes during the quarter. The bold line in each figure connects the centroid of the triangle with the front apex. As the swimmer's mass is evenly distributed to nodes and scales on all sides, the centroid is also the swimmer's centre of mass. Over a larger turn, the swimmer will tend to drift, but it can be seen here that for a single oscillation, the position of the centre of mass is approximately constant.

and everything else behind it, but also to be able to change direction on-demand in a dynamic and unpredictable environment.

In explaining their theory of purposeful behaviour, Rosenblueth and Wiener (1950) use the example of a dog race, where biological and mechanical hounds chase after a mechanical hare. They argue that the behaviour of the hare cannot be considered purposeful, as if it is pushed off course it will not return to its original trajectory, but this will not do for the hounds. If the hare changes course, then the hounds will also change course, otherwise they would fail in their attempts to chase and/or catch the hare (the purpose of winning the race belongs to the hounds' owners, and is only achieved through exploitation of the hounds' intrinsic motivations). In robotics and artificial life alike, the subjects of locomotion studies tend to be more like the hare in this story than the hounds. Technically, this usually manifests in the absence of exteroceptive feedback in the controller. As the cyberneticians realised, it is not possible to achieve stability of behaviour and its results without closing the sensorimotor loop, and - as in the work presented here - it is to the hounds that we should turn for our example when we study locomotion.

The early chapters of Braitenberg's *Vehicles* (Braitenberg, 1986) are a study of how simple sensorimotor couplings may implicitly embody goals and generate purposeful behaviour, in an appropriate environment. Braitenberg's vehicles move and steer through the use of differentially driven wheels. However, while examples of rolling locomotion (including those in (Full et al., 1993; Brackenbury, 1997; García-París and Deban, 1995)), and even wheels of sorts (Gould, 1983), are known in animals, differentially driven wheels are not, and as earlier noted by Keijzer (1998), the use of wheels may be one abstraction too far from an animal's body, if we wish to study brain-body-environment interactions in the tradition of embodied cognitive science. Therefore we extended Braitenberg's model to include a simple deformable body, which can produce behaviours like his vehicles with little more than the same sensorimotor couplings. The main difference here, in control terms, arises from the realisation that for most locomoting morphologies it is not enough to simply excite the actuated elements, but they must be excited in a way which is spatiotemporally coordinated. For wheeled locomotion, often only the velocities of the wheels are pertinent - in other words, frequency of rotation is all important and phase may be neglected, but this is not the norm for walking, crawling, flying, or swimming.

Keijzer et al. (2013) have characterised Braitenberg's vehicles as input-output (IO) machines, a theoretical foundation for nervous systems which they contrast with the internal coordination (IC) model, in which the internal coordination of behaviour is considered a more fundamental function. However, in a subsequent paper, Jékely et al. (2015) point out that many animals' behaviours require a combination of IO and IC in the nervous system. They also refer to an interesting example of coordination which does not arise from a nervous system, in a ctenophore comb (Tamm, 1984), where beating cilia are synchronised by hydrodynamic coupling. This is an example which some would also classify as morphological computation, or possibly morphological control (Füchslin et al., 2013; Müller and Hoffmann, 2017). We propose the alternative appellation 'morphological coordination', as opposed to neural coordination, and will add two more supporting examples. In the case of the aforementioned passive dynamic walker, it has no controller and so we can easily dismiss control from our analysis. It is also difficult to defend the idea of computation per se in this case (Müller and Hoffmann, 2017), although this issue may not yet be closed. We suggest that it is more intuitive to describe the properties of the walker's morphology that contribute to its walk as morphological coordination. The spatial distribution of its joints and its mass, with the assistance of a gentle incline and the force of gravity, naturally give rise to the dynamic pattern of one foot following the other; in other words, not only do they make the walker head downslope, but they also determine the relative phases of motion of the legs. In a biological analogy, it has been shown that a dead rainbow trout can swim forwards through the wake of an object in front of it (Beal et al., 2006). Once again, as this is a dead fish and therefore lacking neural activity, there is no controller here and so the presence of control or computation are both problematic to defend in this context (but again, this may not be the final word). Internal morphological coordination,

arising from the passive dynamical properties of the fish, is an easier way to describe what is happening here. In fact, the remarkable thing about the swimming of the dead fish is that, like the passive dynamic walker, it is not only not controlled but also not internally powered: it swims forwards by passively extracting more energy from the turbulent flow around it than it loses to drag from the surrounding fluid. Both the fish and the walker achieve their observed actions through the spatiotemporally coordinated absorption, storage, and release of energy.

Simple as it is, our swimmer does not trivialise the problem of locomotion through the use of wheels, and so must incorporate both characteristics: input-output in the form of connections like Braitenberg's from sensor to motor, but also internal coordination of muscular activity from the pacemaker. We can compare the contrast between IO machines and the IC model to that between reflex pattern generators (RPGs) and central pattern generators (CPGs). Both seem common in biology, although it is not always easy to identify which mechanism is at work in a particular behaviour (Hoinville et al., 2015). Our motor controller, combining a pacemaker with phase and amplitude modulation due to sensory input, combines IC and IO, similarly to a mixed pattern generator (MPG), a hybrid between central and reflexive pattern generation (Beer, 2009). In addition, there is a third factor at work here, which is also present in a similar form in Braitenberg's vehicles. In those, although each sensorimotor loop is an independent input-output subsystem, they are positioned and aligned in such a way as to correctly coordinate the vehicles' behaviours. This is most easily explained by returning to an essentially cybernetic analysis. When we look at the output of the motors of a Braitenberg vehicle, they rise as the vehicle approaches a source of stimulus. In other words, the individual sensorimotor loops are not stable, and we can locate no behavioural goal at this level. However, if we take the difference between the measurements taken at both sensors as our controlled variable, then we can broadly view the overall system as controlling this variable with negative feedback, and a zero-valued reference point. In one sense, this is the goal of our system of interest, and at this level the same goal applies to both positive and negative phototaxis - the vehicle moves in such a way as to equalise the sensory measurements and in doing so either steers towards or away from the stimulus source. Which behavioural goal is implicit in the vehicle depends on whether the connections are straight or crossed, but both behaviours rely on the correct spatial relationships between sensors and motors, an example of what we have termed morphological coordination. If a vehicle's sensors and motors are arranged in a way which is far from symmetrical, then these behaviours are not assured. For example, if both sensors are placed on one side of the vehicle, then it is possible for a vehicle to get trapped in an orbit where the sensor measurements never equalise.

As in the robotic examples of (Shintake et al., 2011; Ziegler et al., 2006), the morphology of our swimmer is such that it can swim under the control of a simple sinusoidally oscillating control signal, and be steered by the modulation of the parameters of that sine wave. However, while those robots can be externally controlled to steer towards or away from targets, in contrast to our swimmers their behaviour does not arise from implicit goals, as their sensorimotor loops are not closed. Salumäe et al. (2012) have closed the loop, and demonstrated that a Braitenberg style controller can be used to enable a fish robot to maintain its alignment to the surrounding water flow. We believe that, in the general study of locomotion, that loop should always be closed, and not just at the level of controlling individual joint positions. Locomotion is an activity which is always directed towards some behavioural purpose, and while there is often value in studying it as an isolated action, it is erroneous to consider it a behaviour in its own right. As is stressed in embodied cognitive science, and its offshoot morphological computation, the roots of purposeful and intelligent behaviour are not found only in nervous systems, or their artificial analogies. As is strongly evidenced by the dead rainbow trout, even in the neuromuscular animals some degree of the coordination of behaviour arises directly from the passive morphological properties of an animal's body. In terms of metabolic cost, nervous systems are not cheap (Niven and Laughlin, 2008), and it is becoming increasingly clear that evolution is not biased towards employing them for functions which bodies can already perform well. This being the case, it remains to discover

what those functions may be, and how far they they may originate and/or pervade more complex behaviours.

Although the 'muscles' of our swimmer do not display the passive energy-storage dynamics evident in the dead rainbow trout, every part of its exterior is complicit in its motion, and as its dimensions are dynamically reconfigured, the dynamics of the surrounding water also conspire towards the swimmer's course to its goal. In a fluidic environment, the importance of morphology is of more clear and central importance than in animals which walk and run. The triangular exterior of our swimmer is a streamlined form; without this property, a more complex morphology might well be required for success, if not also a more complex control regime.

The dynamics which give the swimmer translational motion arise from the interactions between its scales and the particles of the surrounding water. To the best of our knowledge, this is the first use of a particle-based fluid simulation with virtual creatures. Particle physics opens up opportunities for the exploration of a wider range of fluid dynamics and also, therefore, a wider range of swimming types than the simple model of Sfakiotakis and Tsakiris (2006), but they are computationally expensive in comparison. For this reason, we have also developed a means of minimising the number of particles required to simulate swimming. There is room for improvement in our 'particle cloud'; for example, a square cloud with particles incoming parallel to the x and y axes is easy to implement, but has a visible effect on the dynamics of certain trajectories. A circular bubble with particles approaching from all angles may neutralise these effects, and perhaps a cloud with a shape which is more fitted to the proportions of the swimmer and its motions will enable a further reduction in the number of simulated particles. The use of particles also necessitated a new means of simulating a 'pseudo-soft' body. We implemented this through the use of rigid body scales, which overlap and slide past each other without collision. This method can be easily extended to more complex triangulated bodies, so that soft-bodied swimmers with hydrostatic skeletons can be constructed and simulated in particle-based fluids.

We have developed a way of using a two-dimensional physics engine to simulate virtual creatures which swim with increased realism over the simple equations often used in related work, but without incurring the higher computational costs associated with more thorough computational fluid dynamics methods. With this platform and the theoretical underpinnings of morphological computation and morphological coordination, we can continue to extend our knowledge of how far the body *can be* implicit in purposeful and intelligent behaviours, which will give us new pointers on what to look for in the analysis of the importance of morphology to animal behaviour.

Chapter 6

Future work: evolving swimming virtual creatures

We have seen from Hauser et al. (2011, 2012) that MSD networks are effective for analog computational applications like filtering and central pattern generation, and from our own results described in part one of this thesis we can add to the list the control of purposeful and intelligent behaviour. The chief reason these networks are of interest is because they can be used to model the bodies of soft robots and animals. Because of this connection, we can hypothesise that the bodies of soft robots and animals are capable of performing the same functions. To some extent, this has already been shown to be the case, by the octopus arm model based on MSDs from Nakajima et al., and with related simulated and real robots (Zhao et al., 2013; Caluwaerts et al., 2014; Eder et al., 2017). However, this has not yet been done with the kinds of active perception and decision making behaviours we were able to achieve with our controllers. We now look to the evolution of virtual creatures with MSD network bodies to make that next step.

6.1 Introduction

The evolution of virtual robots and creatures has led to important insights and hypotheses into the role of morphology in behaviour. Notable examples include the observation that coevolving of a controller and body can enhance evolvability (Bongard, 2010), support for the hypothesis that a flexible spine enhances efficiency in quadrupedal locomotion (Moore et al., 2015), and a study of the relationship between complexity in the environment and the morphological complexity of the bodies evolved to locomote in it(Auerbach and Bongard, 2014). The use of a virtual environment for asking questions like these has a number of advantages. Some are related to practical concerns such as accessibility and cost, but from the scientific point of view the chief advantages are having complete control over experimental conditions, and having complete access to the state of all simulated entities.

We are interested in evolving swimming virtual creatures for two main reasons: first, because a soft and fluid environment offers rich variety in body-environment interactions, and second because early neuromuscular systems evolved in the sea. As it is often easier to identify morphological computation and coordination in simpler bodies and behaviours, so it might be that these characteristics played a more prominent role in the behaviour of early swimmers than they do in many animals today. In Chapter 5, we showed a simple example of morphological coordination in a soft-bodied swimmer, which was designed and controlled to behave similarly to Braitenberg's vehicles (Braitenberg, 1986). One problem we have to solve for coevolving more complex bodies and controllers is this: how do we connect a controller such as a neural network to an arbitrary morphology? In recent virtual creatures with MSD network bodies, with coevolved controllers, Schramm and Sendhoff (2011) attached central pattern generators which drive the contraction of adjacent springs, and Joachimczak et al. (2016) used similar but more abstract oscillators to drive

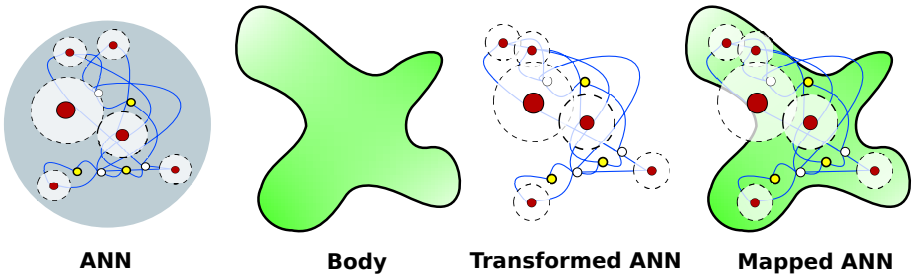


Figure 6.1 The principle of mapping an ANN to an arbitrary body morphology. An ANN and a body morphology are separately encoded, and initially generated in their own coordinates. Then the ANN coordinates are transformed to lie within the body. Finally, the ANN is attached to the body.

contractions for each modelled cell in the bodies of their creatures. Although the authors of these works were not focussed on our themes of morphological computation and coordination, we believe their methods are ripe for studies which are. However, we hypothesise that such simple controller designs may be limited to relatively simple behaviours, and are developing a method for the attachment of arbitrarily complex neural networks to arbitrary MSD network morphologies, which we will describe in the following sections.

6.1.1 Attachment and actuation

In our approach, all the nodes in an artificial neural network attach to either nodes or springs in an MSD network body (Figure 6.2). Neuron types fall into three classes, sensor neurons, interneurons, and motor neurons. A sensor neuron may proprioceptively detect local body state, or some exteroceptive stimuli in the environment. An activated motor neuron causes the springs in its vicinity to contract. As neurons have position but no size, and not all potentially active elements of the body need be actuated, a neural network of any size can be attached to any body, and so the complexities of body and neural network morphologies are decoupled. For example, a highly complex morphology may be controlled by a simple neural network which generates a periodic control signal for a single motor neuron, which in turn actuates a large region of the body. Similarly, a relatively simple body can be controlled by a very large neural network which performs complex processing of sensory streams.

6.1.2 Mapping ANNs to bodies

As well as enabling flexibility in the relative scales and complexities of body and controller, our approach also enables great flexibility in how the two components are evolved. This is achieved by evolving the ANN and the body in separate spaces, and then mapping the ANN to the body at the point when a virtual creature phenotype is constructed (Figure 6.1). This means that, if desired, different encodings and even different evolutionary algorithms may be used for the body and the ANN, as long as they proceed together.

The body and the neural network are encoded separately, and are initially constructed in their own spaces. The boundary of the body is its profile, and the boundary of the neural network is a circle which envelops it. The centre and radius of the circle are defined as follows: The centroid (a coordinate which is the average of all coordinates of nodes in the neural network) of the neural network is selected as the centre of the circle. Then, the maximum distance from centroid to neural network nodes is found, and the radius of the circle is defined as that distance multiplied by 10/9, so that there is an empty border region around the neural network which is proportional to its extents.

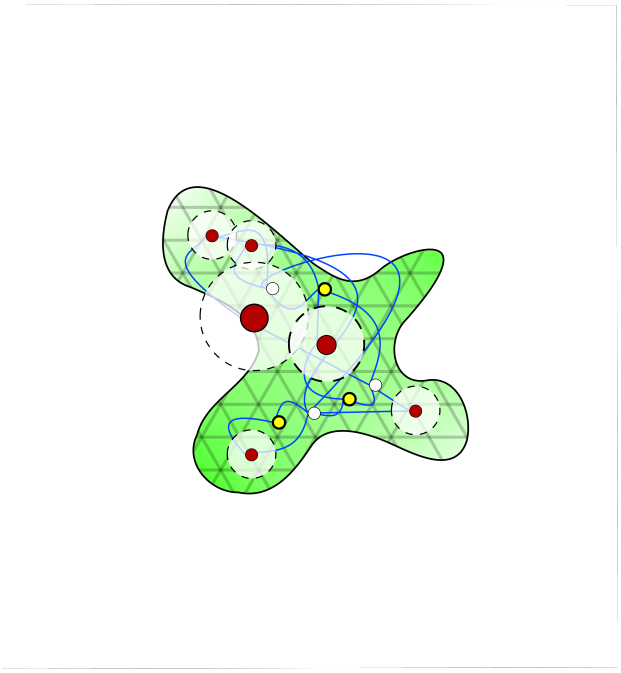


Figure 6.2 An ANN attached to a triangulated body. The nodes in the ANN are attached to the nearest nodes or joints to their position, and motor neurons activate all joints within a prescribed radius from their position. Spheres of influence of motor neurons may overlap, in which case joints affected by more than one motor neuron will be activated according to a linear combination of the motor neuron's outputs.

The process of mapping the neural network to the body has two phases. The first is to determine a mapping between the two boundaries. The edge of the body is found, approximately, using image processing techniques. First, the body is drawn as a silhouette, black on a white background. The pixels on the edge of the silhouette are found by searching for all pixels which are both black and adjacent to one or more pixels that are white. Then, the list of those pixels is sorted into an order that traverses the body's boundary moving from each edge pixel to its next edge neighbour. Each pixel is then assigned what we might call a pseudo-angle, a value in the interval [0, 2PI), according to its position in the list. Points on the circular boundary are assigned to the same interval, in the conventional way, as angles between the vector from centroid to boundary point and the x-axis. The vector of pseudo-angles is then rotated such that it most closely matches the vector of angles on the circle.

The second phase is a relaxation algorithm which uses these two edges, and the correspondences between angles and pseudo-angles, to deform the neural network plane (Figure 6.3). A number of control points, on the perimeter of the circle, are selected. The corresponding points on the body's edge are selected according to the closest correspondence between the angle for the control point, and the pseudo-angle of the body's edge coordinates. Then, using the Box2d physics engine, a circular mesh of mass-spring-dampers in a triangulated configuration is constructed using the control points. The coordinates of all nodes in the neural network are adjusted such that they attach to vertices in the mesh. When the plane is deformed, those vertices can then be used to directly determine the transformed coordinates of the nodes.

The plane is deformed using the following procedure. The Box2d simulation is run, and on each step, which has a duration of 1ms, the control nodes of the mesh are pushed towards their corresponding points on the body's edge. After the nodes are pushed, the rest lengths of all springs in the network are then updated to be equal to their current lengths, so that the network is stabilised. This ensures that the mesh resists deformation, so that it does not collapse or distort uncontrollably, but without the stiffness forces from the springs growing so much as to make the mesh unstable

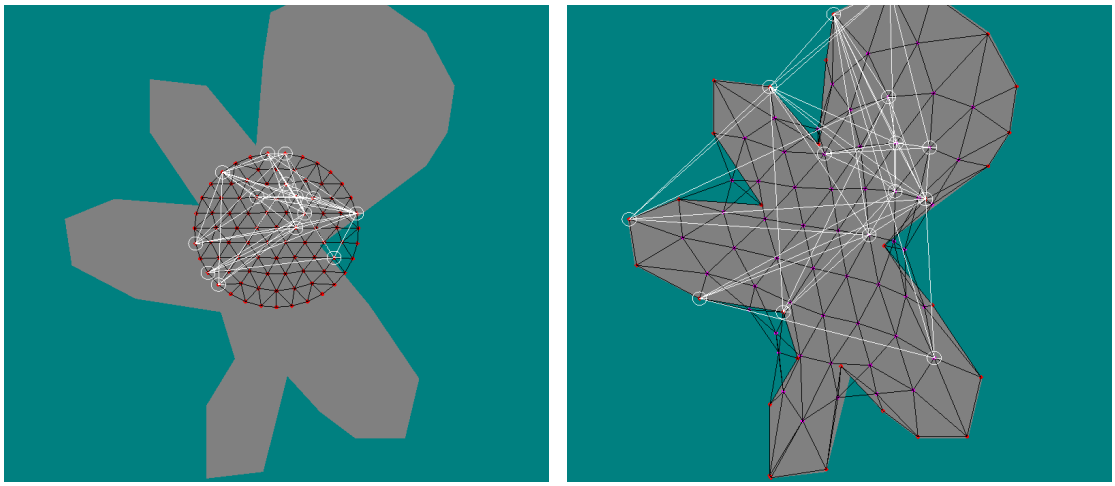


Figure 6.3 Mapping a neural network (white) to a body plan (grey). The neural network is first attached to a circular mesh. The mesh is then deformed until it is aligned with the body, so that the neural network is mapped to the body's space. The current relaxation algorithm is imperfect, and during relaxation parts of the mesh can fold over each other, leaving some nodes of the mesh outside of the body's profile.

and require massive forces to compress. Ideally, as long as a sufficient number of control points are selected, the mesh will be deformed to the point where all of its vertices are contained within the profile of the body. The locations of the transformed neural network nodes in body space are then taken as the coordinates of their corresponding vertices.

6.1.3 Mutations

As shown in figures 6.4 and 6.5, when a body morphology changes, the ANN is remapped accordingly. This can have various effects. Although the internal connections of the ANN do not change, the distances between them do, which can have an effect on network dynamics if the network wires implement a transport delay which is proportional to wire length. Changes in body morphology and mapping can also have a large effect by repositioning sensor and motor neurons on the body.

6.1.4 Alternative mappings

We are also developing a second method for mapping from the space of a neural network to that of a body, which is in principle simpler than the one described above, but potentially more sophisticated and easier to apply to a three-dimensional format.

In this method, the neural network is evolved in a polar coordinate system. For each given neuron, we then have an angle and a radius. We use the radius to generate a corresponding contour inside the body, and then map from the neuron's circle to the body contour using the same method described above (Figure 6.6).

One important advantage of this mapping method over the relaxation method we previously described is that it simpler to extend it to three dimensions. In the three dimensional case, the neural network can be evolved in spherical coordinates, and for mapping purposes the body can be treated as the surface of a three-dimensional volume. The origin of the sphere is placed at the body's centre of mass. The first angle from a neuron's spherical coordinate then specifies the angle in the xy-plane of a cross-section through the body in the yz-plane. Once we have the profile that cross-section produces, the problem reduces to the same polar coordinate mapping that we described above.

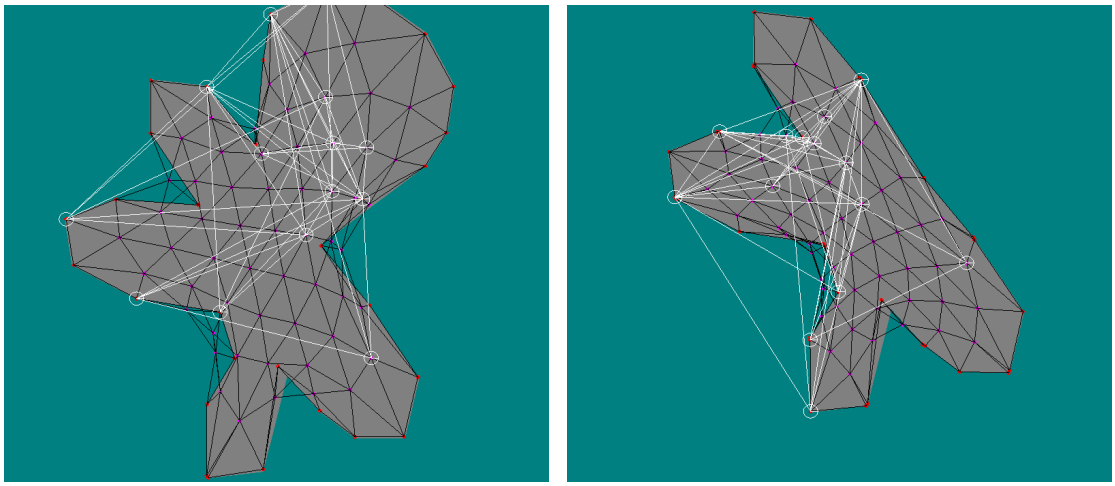


Figure 6.4 A change in morphology can lead to large changes in the way the neural network attaches to the body, but internally the neural network itself has the same connections and properties. However, if the length of wires (in body space) connecting the nodes of the neural network is used to set transport delays, then the dynamics of the network may still be modified.

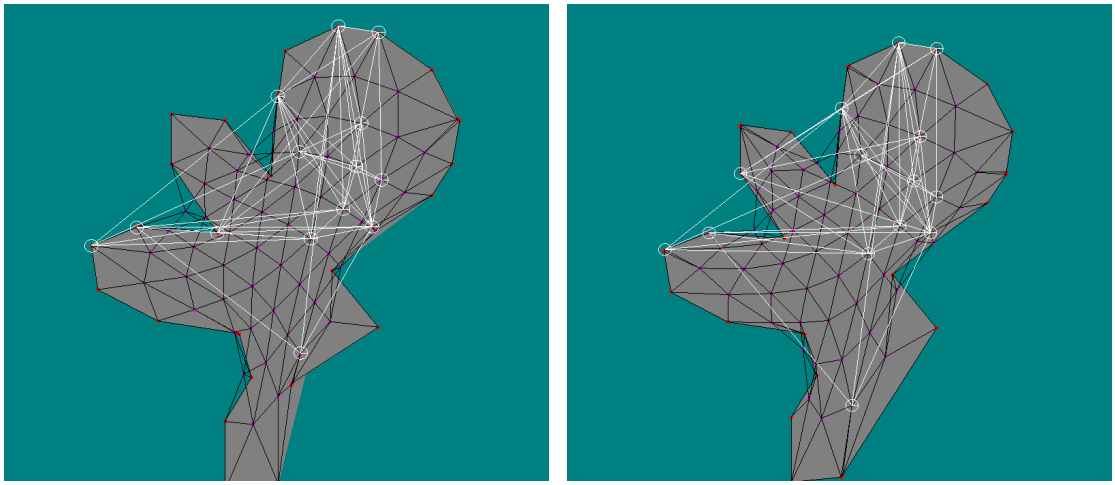


Figure 6.5 A change in morphology which leads to relatively small changes in the neural network connections to the body. Interestingly, although the overall difference seems relatively small, changing the morphology in the lower right-hand quadrant has led to a change in the opposite quadrant, as one node has moved from inside the body to the upper left appendage.

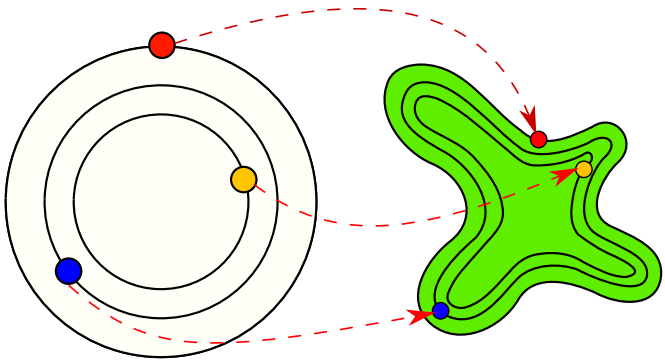
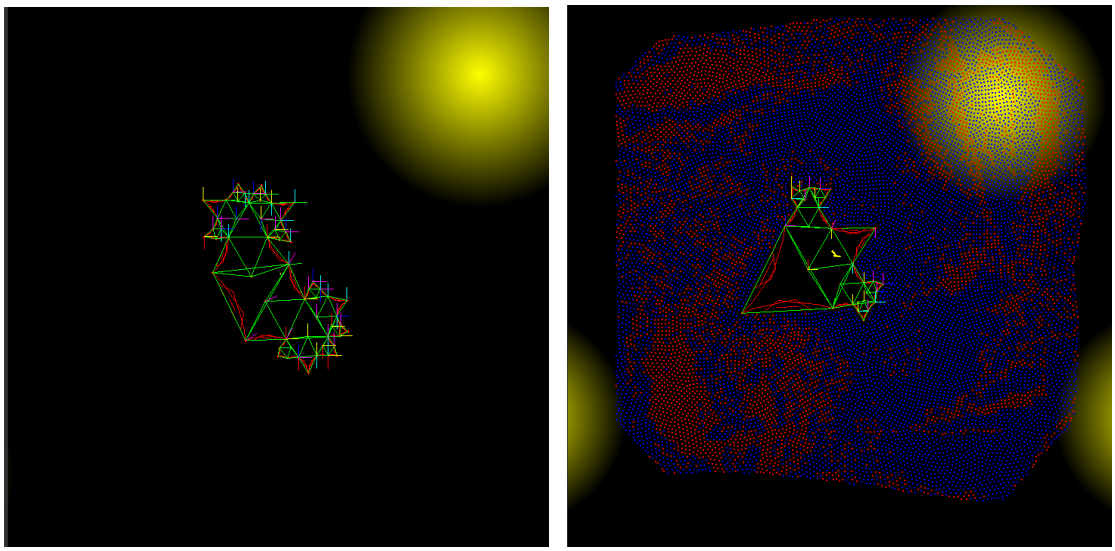


Figure 6.6 A mapping method based on contours. A neural network is evolved in polar coordinates. For every neuron in the network, a circle can be drawn. For every circle, a corresponding contour can be generated for the body to map to. Once this is done, it is simple to place a neuron on that contour.



(a)A swimming virtual creature in motion. The swimmer's body is constructed according to a grammar-based approach with an evolved set of production rules. The body is essentially a mass-spring-damper network, with overlapping scales around its exterior, and the swimmer moves by contracting individual mass-springer-dampers.

(b)A swimming virtual creature, surrounded by three chemical sources and with its particle cloud shown.

Figure 6.7 Future swimmers. Similarly to those of Joachimczak et al. (2016), these swimmers are powered by simple sensorimotor circuits on all active elements, but it is unclear how complex the emergent behaviour will be. We believe our mapping method may make complex behaviour easier to evolve.

6.2 Conclusion

In this chapter we have introduced a new concept for the coevolution of brains and bodies, targeted at, but not exclusive to, our future work with MSD network virtual creatures. While there are numerous examples of novel forms and interesting behaviour in virtual creatures (including (Sims, 1994; Cheney et al., 2014; Joachimczak et al., 2016; Corucci et al., 2018)), it has proven challenging to achieve complex behaviours (Lessin et al., 2013; Lipson et al., 2016). We believe that the flexibility granted to evolution by this method, in terms of decoupling the encodings of neural network and body, may lead to more complex virtual creatures than have previously been reported.

This method is one option amongst others for the control of our MSD bodied swimmers. Other methods which we are developing include giving all active elements their own simple sensorimotor circuits(Figure 6.7), and the physical reservoir computing approach which is one of the central themes of this thesis. Behaviour is a product of interactions between controller, body and environment (Chiel and Beer, 1997). Morphological computation and coordination are predominantly focussed on interactions between body and environment, but for more complex behaviours, these interactions in turn interact with complex controllers, and so we are making room for such controllers in our framework.

Conclusion

In Chapter 1, we began by reviewing the story of morphological computation so far: the introduction and development of the underlying theory, important results from robotics, and the physical reservoir computing approach. As we saw, morphological computation as a whole is not precisely defined, and there have been few attempts to remedy this situation. Ultimately, we concur with Müller and Hoffmann (2017) when they argue that in what is often referred to as morphological computation, only physical reservoir computing may be genuinely computational. If the greater part of morphological computation is not genuinely computational, then how should it be described? Füchslin et al. (2013) introduce the idea of morphological control, but as they describe this as 'control achieved via morphological computation', they do not answer our question. Müller and Hoffmann go part way when they explain morphological control as 'morphology facilitating actuation'. However, there are examples, such as the passive dynamic walker and the dead rainbow trout (Chapter 5), which clearly have much in common with morphological computation and control, but which lack controllers altogether. For examples such as these, we do not believe that theories of either computation or control fit, and we suggest that in these cases morphology is better viewed as coordinative of behaviour. In one sense, morphological coordination might be viewed as a physical analogy to the *internal coordination* of nervous systems (Keijzer et al., 2013). Another way of looking at morphological coordination is that it describes the *potentialities* of a body. We believe we have only scratched the surface of this idea in this thesis, and see it as a rich seam which can be mined for explaining aspects of behaviour. According to Carson and Kelso (2004), 'The coordination of movement is governed by a coalition of constraints.' For a hyper-redundant body like that of a human, for example, it is not possible to exhaustively list all the things it may do. In contrast, part of the reason the passive dynamic walker can locomote without control is because its dynamics are so constrained by its morphology, and consequently it is good for nothing else. The trout, on the other hand, lies somewhere between these two extremes. It has more degrees of freedom than the passive dynamic walker, and more constraints, but they are also weaker than those of the walker. Some of those constraints, morphological and dynamical, come to the fore after the fish dies, and cause it to act like an automaton under the right conditions.

The other main insight that we gained from Chapter 1 is that the emphasis in physical reservoir computing has been on computational tasks like filtering, and on controlling what we might call behavioural primitives, such as straight-line locomotion. We also noted that where physical reservoir computing had been applied to robotics, the reservoir was always made up of only proprioceptive sensors.

Our intuition was that the dynamics of MSD networks might be applied to controlling a mobile agent engaged in more challenging behavioural tasks, and that they might be 'programmed' via an evolutionary algorithm to respond appropriately to exteroceptive sensory inputs applied as forces to the passive compliant reservoir. In Chapter 2 we were able to demonstrate that our hypothesis was correct, and that small and simple MSD network controllers could perform well in solving a behavioural problem which has previously been referred to as *minimally cognitive*. As successful as these controllers were, we recognised that the behavioural challenge was simple in some important aspects: firstly, the controller only had to move an animat left and right along a single axis, and secondly, it operated in a noise free world. In addition, it appears that the problem can be solved with purely reactive controllers and does not in fact require memory in the controller.

That being the case, in the work reported on in Chapter 3 we set our controllers a more demanding challenge. Not only did we pick a task which explicitly required memory, but we also added a second dimension to the environment, with the controller directing a simulated robot in a two-dimensional maze. We also introduced noise and some random variation in the dimensions of the maze. In this case, even after many design iterations and applying a number of evolutionary tricks, we still only obtained a few controllers which had a high success rate in solving the given problem. In the end, we can speculate about why exactly this is a much harder task for our controllers than the one in Chapter 2, but the important result is that any successful controllers were evolved at all. We didn't stop work on this problem because we had run out of ideas, but rather because we eventually decided that we had devoted enough time to our attempts. It may be that not much needs to be altered in our design to finally crack the problem. While we spent time working on this memory-related problem and controller designs, we developed a novel persistent bistable memory element for MSD networks, which could be used as a separate but connected memory element for use with MSD network reservoirs, or could be built into a network which is used as a reservoir.

While the first part of this thesis is focussed on the properties and uses of MSD networks as controllers for behaving bodies, in the second part of the thesis we describe the beginnings of the next step in our research, which is to use MSD networks as the bodies that behave. Chapter 4 draws a dotted line between the two parts of the thesis as it describes an MSD network being used as an analogue computing device, but with an important modification. We changed the mode of input to the network from external forces operating on a passive compliant network to internal forces which drive the network directly by actuating MSDs. We were interested to see what the effect of this would be, as our virtual creatures in Chapters 5 and 6 are actuated in the same way. We discovered that the performance of the networks was actually slightly improved by this change, for the given task of emulating a Volterra series operator, and also that performance only degraded gradually when we varied the size of the network and the proportion of MSDs which were used as inputs to the system. We were encouraged by these results as they indicate that the MSD network bodies of our virtual creatures could be used for such challenging computational tasks while they are in motion. For example, we could train linear readouts for closed-loop motor control with a straightforward reservoir computing approach, or we could control our virtual creatures with other types of neural network in combination with linear readouts which perform complex sensory preprocessing.

For our virtual creature experiments, we have elected to evolve swimmers in a watery environment. We find behaviour in fluidic environments an appealing subject because there is the potential for many kinds of interaction between the body and the environment when both are soft. Existing approaches to swimming virtual creatures have employed a simple fluid drag model (Sfakiotakis and Tsakiris, 2006), in which the interactions between bodies and water incorporate simple forces, but fail to include features such as viscosity and turbulence. For this reason, we developed our own methods for using Google's *liquidfun* (Google, 2013) particle-based physics engine to simulate what we call *pseudo-soft*-bodied swimmers in particle-based fluids, methods which we describe in Chapter 5. The computational cost of particle-based physics can become prohibitive, so we developed a novel method for minimising the number of particles by only simulating the water in the vicinity of the body, with a kind of moving water tunnel. We solved the problem of interactions between our MSD network bodies and the surrounding fluid by enclosing our swimmers with a skin of rigid body overlapping scales, which slide over one another to allow deformations of the body.

We are currently developing a framework for evolving more complex behaviours and bodies for our MSD network swimmers. In Chapter 6 we describe what we believe to be an important innovation that has emerged during that development: methods for mapping arbitrary artificial neural networks for sensorimotor behaviour to arbitrary body morphologies. In the near future, we will be applying the methods from part two of this thesis to continue our investigation into the role of morphological computation and coordination in behaviour using two-dimensional MSD

networks. In the long term, we intend to extend these methods into three dimensions.

Bibliography

Albert, J. (1999). Computational modeling of an early evolutionary stage of the nervous system. *Biosystems*, 54(1):77–90. 55

Ashby, W. R. (1956). *An introduction to cybernetics*. Chapman and Hall, London. 14

Auerbach, J. E. and Bongard, J. C. (2014). Environmental influence on the evolution of morphological complexity in machines. *PLOS Computational Biology*, 10(1):e1003399. 54, 72

Baratta, R. and Solomonow, M. (1990). The dynamic response model of nine different skeletal muscles. *Biomedical Engineering, IEEE Transactions on*, 37(3):243–251. 14

Barnett, L. (1998). Ruggedness and neutrality-the nkp family of fitness landscapes. In Adami, C., Belew, R. K., Kitano, H., and Taylor, C. E., editors, *Artificial Life VI: Proceedings of the sixth international conference on Artificial life*, pages 18–27. 21

Beal, D., Hover, F., Triantafyllou, M., Liao, J., and Lauder, G. (2006). Passive propulsion in vortex wakes. *Journal of Fluid Mechanics*, 549:385–402. 69

Beer, R. D. (1996). Toward the evolution of dynamical neural networks for minimally cognitive behavior. In Maes, P., Wilson, S. W., and Mataric, M. J., editors, *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 421–429, Cambridge, MA. MIT Press. 1, 13, 15, 18, 23

Beer, R. D. (2003). The dynamics of active categorical perception in an evolved model agent. *Adaptive Behavior*, 11(4):209–243. 15, 23, 25

Beer, R. D. (2009). Beyond control: The dynamics of brain-body-environment interaction in motor systems. In Sternad, D., editor, *Progress in Motor Control*, pages 7–24. Springer. 70

Blynel, J. and Floreano, D. (2003). Exploring the t-maze: Evolving learning-like robot behaviors using ctrnns. In et al., C. S., editor, *Applications of Evolutionary Computing*, volume 2611 of *Lecture Notes in Computer Science*, pages 593–604, Berlin, Heidelberg. Springer. 1, 38, 39

Bongard, J. (2010). The utility of evolving simulated robot morphology increases with task complexity for object manipulation. *Artificial Life*, 16(3):201–223. 54, 72

Bongard, J. (2014). Why morphology matters. In Vargas, P. A., Di Paolo, E. A., Harvey, I., and Husbands, P., editors, *The Horizons of Evolutionary Robotics*, chapter 6, pages 125–152. MIT Press, Cambridge, MA. 21

Brackenbury, J. (1997). Caterpillar kinematics. *Nature*, 390(6659):453–453. 69

Braitenberg, V. (1986). *Vehicles: Experiments in synthetic psychology*. MIT press, Cambridge, MA. 13, 54, 69, 72

Brooks, R. A. (1991). Intelligence without representation. *Artificial intelligence*, 47(1):139–159. 13

Caluwaerts, K., Despraz, J., Işçen, A., Sabelhaus, A. P., Bruce, J., Schrauwen, B., and SunSpiral, V. (2014). Design and control of compliant tensegrity robots through simulation and hardware validation. *Journal of The Royal Society Interface*, 11(98):20140520. 48, 72

Caluwaerts, K., D'Haene, M., Verstraeten, D., and Schrauwen, B. (2013). Locomotion without a brain: Physical reservoir computing in tensegrity structures. *Artificial Life*, 19(1):35–66. 10, 55, 56

Cannon, W. B. W. B. (1953). *Bodily Changes in Pain, Hunger, Fear and Rage : an Account of recent Researches into the Function of Emotional Excitement*. Charles T. Branford Co, 2 ed.. edition. 37

Carson, R. G. and Kelso, J. A. S. (2004). Governing coordination: behavioural principles and neural correlates. *Experimental Brain Research*, 154(3):267–274. 78

Catto, E. (2013). Box2d: A 2d physics engine for games, *available at www.box2d.org*. 55

Chen, J.-C. and Conrad, M. (1994). A multilevel neuromolecular architecture that uses the extradimensional bypass principle to facilitate evolutionary learning. *Physica D: Nonlinear Phenomena*, 75(1-3):417–437. 21

Cheney, N., Clune, J., and Lipson, H. (2014). Evolved electrophysiological soft robots. In Sayama, H., Rieffel, J., Risi, S., Doursat, R., and Lipson, H., editors, *ARTIFICIAL LIFE 14: Proceedings of the Fourteenth Conference on the Synthesis and Simulation of Living Systems*, pages 222–229, Cambridge, MA. MIT press. 55, 58, 77

Chiel, H. J. and Beer, R. D. (1997). The brain has a body: adaptive behavior emerges from interactions of nervous system, body and environment. *Trends in Neurosciences*, 20(12):553–557. 77

Clark, A. (1996). *Being there: Putting brain, body, and world together again*. MIT press. 53

Collins, R. J. and Jefferson, D. R. (1991). Selection in massively parallel genetic algorithms. In Belew, R. K. and Booker, L. B., editors, *Proceedings of the Fourth International Conference on Genetic Algorithms, ICGA-91*, pages 249–256. Morgan Kaufmann. 41

Corucci, F., Cheney, N., Giorgio-Serchi, F., Bongard, J., and Laschi, C. (2018). Evolving soft locomotion in aquatic and terrestrial environments: Effects of material properties and environmental transitions. *Soft robotics*, 5(4):475–495. 77

Corucci, F., Cheney, N., Lipson, H., Laschi, C., and Bongard, J. (2016). Evolving swimming soft-bodied creatures. In *ALIFE XV, The Fifteenth International Conference on the Synthesis and Simulation of Living Systems, Late Breaking Proceedings*, pages 6–7. 55

Coumans, E. (2016). Bullet physics library, *available at http://bulletphysics.org/wordpress/*. 55

Dale, K. and Husbands, P. (2010). The evolution of reaction-diffusion controllers for minimally cognitive agents. *Artificial Life*, 16(1):1–19. 15

Diamond, A. (2013). *Bio-inspired approaches to the control and modelling of an anthropomimetic robot*. PhD thesis, University of Sussex. 55

Doursat, R. and Sánchez, C. (2014). Growing fine-grained multicellular robots. *Soft Robotics*, 1(2):110–121. 55

Eder, M., Hisch, F., and Hauser, H. (2017). Morphological computation-based control of a modular, pneumatically driven, soft robotic arm. *Advanced Robotics*, pages 1–11. 10, 48, 72

Feynman, R. P. (2005). *Perfectly reasonable deviations from the beaten track: The letters of Richard P. Feynman*. Basic Books. xi

Füchslin, R. M., Dzyakanchuk, A., Flumini, D., Hauser, H., Hunt, K. J., Luchsinger, R. H., Reller, B., Scheidegger, S., and Walker, R. (2013). Morphological computation and morphological control: Steps toward a formal theory and applications. *Artificial Life*, 19(1):9–34. 10, 11, 54, 69, 78

Full, R., Earls, K., Wong, M., and Caldwell, R. (1993). Locomotion like a wheel? *Nature*, 365:495. 69

García-París, M. and Deban, S. M. (1995). A novel antipredator mechanism in salamanders: Rolling escape in hydromantes platycephalus. *Journal of Herpetology*, 29(1):149–151. 69

Germann, J., Maesani, A., Stockli, M., and Floreano, D. (2013). Soft cell simulator: A tool to study soft multi-cellular robots. In *Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference on*, pages 1300–1305. IEEE. 55

Geyer, H., Seyfarth, A., and 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. 14

Goldschlager, L. and Lister, A. (1982). *Computer Science: A Modern Introduction*. Prentice Hall International (UK) Ltd., Hertfordshire, UK. 14

Google (2013). liquidfun, *available at http://google.github.io/liquidfun/*. 2, 43, 55, 79

Gould, S. J. (1983). *Hen's Teeth and Horse's Toes: Further Reflections in Natural History*. W. W. Norton and Company. 69

Harvey, I. (1997). Cognition is not computation; evolution is not optimisation. In W., G., A., G., M., H., and D., N. J., editors, *International Conference on Artificial Neural Networks*, volume 1327 of *Lecture Notes in Computer Science*, pages 685–690, Berlin, Heidelberg. Springer. 11

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. 1, 8, 9, 13, 14, 17, 18, 19, 45, 46, 47, 54, 72

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. 9, 13, 14, 30, 43, 72

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

Hiller, J. and Lipson, H. (2014). Dynamic simulation of soft multimaterial 3d-printed objects. *Soft Robotics*, 1(1):88–101. 55

Hoinville, T., Schilling, M., and Cruse, H. (2015). Control of rhythmic behavior: central and peripheral influences to pattern generation. In *ICRA 2015 CPG Workshop: CPGs for Locomotion Control: Pros, Cons & Alternatives*. 70

Husbands, P. (1994). Distributed coevolutionary genetic algorithms for multi-criteria and multi-constraint optimisation. In Fogarty, T. C., editor, *Evolutionary Computing, AISB Workshop Selected Papers*, volume 865 of *Lecture Notes in Computer Science*, pages 150–165, Berlin, Heidelberg. Springer, Springer. 41

Husbands, P. (1998). Evolving robot behaviours with diffusing gas networks. In Husbands, P. and Meyer, J.-A., editors, *Evolutionary Robotics: Proc. EvoRob'98*, volume 1468 of *Lecture Notes in Computer Science*, pages 71–86, Berlin. Springer Verlag. 14

Hustig-Schultz, D., SunSpiral, V., and Teodorescu, M. (2016). Morphological design for controlled tensegrity quadruped locomotion. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 4714–4719. IEEE. 55

Iida, F. (2005). Cheap design approach to adaptive behavior: Walking and sensing through body dynamics. In *3rd International Symposium on Adaptive Motion of Animals and Machines*, page 15. Citeseer. 6, 7

Iida, F. and Pfeifer, R. (2004). "cheap" rapid locomotion of a quadruped robot: Self-stabilization of bounding gait. In Groen, F., Amato, N., Bonarini, A. andYoshida, E., and Kröse, B., editors, *Proceedings of the 8th International Conference on Intelligent Autonomous Systems (IAS-8)*, pages 642–649, Amsterdam, Netherlands. IOS Press. 13, 14

Jaeger, H. (2002). *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*, volume 5. GMD-Forschungszentrum Informationstechnik Bonn. 44

Jékely, G., Keijzer, F., and Godfrey-Smith, P. (2015). An option space for early neural evolution. *Philosophical Transactions of the Royal Society B*, 370(1684):20150181. 69

Joachimczak, M., Suzuki, R., and Arita, T. (2016). Artificial metamorphosis: Evolutionary design of transforming, soft-bodied robots. *Artificial Life*, 22(3):271–298. 45, 51, 56, 72, 77

Joachimczak, M. and Wróbel, B. (2012). Co-evolution of morphology and control of soft-bodied multicellular animats. In Soule, T., Auger, A., Blum, C., Branke, J., Bredeche, N., Browne, W., Clark, J., Deb, K., Dorin, A., Doursat, R., Ekart, A., Friedrich, T., Gustafson, S., Hornby, G., Igel, C., Kovacs, T., Landa-Silva, D., Lobo, F., Pappa, G., Lozano, J., Meyer-Nieberg, S., Motsinger, A., Neumann, F., Ochoa, G., Olague, G., Ong, Y.-S., Pelikan, M., Pelta, D., Pizzuti, C., Preuss, M., Rowe, J., Smith, S., Solnon, C., Squillero, G., Tauritz, D., Wong, M., Yoo, S., and Yu, T., editors, *Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation*, pages 561–568, New York, NY, USA. ACM. 55

Johnson, C., Philippides, A., and Husbands, P. (2014). Active shape discrimination with physical reservoir computers. In Sayama, H., Rieffel, J., Risi, S., Doursat, R., and Lipson, H., editors, *ALIFE 14: The Fourteenth Conference on the Synthesis and Simulation of Living Systems*, pages 176–183, Cambridge, MA. MIT Press. 2, 13, 29

Johnson, C., Philippides, A., and Husbands, P. (2015). Maze navigation and memory with physical reservoir computers. In Andrews, P., Caves, L., Doursat, R., Hickinbotham, S., Polack, F., Stepney, S., Taylor, T., and Timmis, J., editors, *Late Breaking Proceedings of the European Conference on Artificial Life 2015*, pages 19–22, Cambridge, MA. MIT Press. 2

Johnson, C., Philippides, A., and Husbands, P. (2016). Active shape discrimination with compliant bodies as reservoir computers. *Artificial Life*, 22(2):241–268. 2, 13, 53

Jones, D. S. (1979). *Elementary Information Theory*. Oxford University Press. 24

Katsuki, T. and Greenspan, R. J. (2013). Jellyfish nervous systems. *Current Biology*, 23(14):R592–R594. 58

Keijzer, F., Van Duijn, M., and Lyon, P. (2013). What nervous systems do: early evolution, input-output, and the skin brain thesis. *Adaptive Behavior*, 21(2):67–85. 69, 78

Keijzer, F. A. (1998). Some armchair worries about wheeled behavior. *From animals to animats*, 5:13–21. 69

Khazanov, M., Humphreys, B., Keat, W., and Rieffel, J. (2013). Exploiting dynamical complexity in a physical tensegrity robot to achieve locomotion. In Liò, P., Miglino, O., Nicosia, G., Nolfi, S., and Pavone, M., editors, *Advances in Artificial Life, ECAL 2013, Proceedings of the twelth European Conference on the Synthesis and Simulation of Living Systems*, pages 965–972. 8

Klyubin, A. S., Polani, D., and Nehaniv, C. L. (2005). Empowerment: A universal agent-centric measure of control. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, pages 128–135. IEEE. 26

Korn, G. A. (2005). Continuous-system simulation and analog computers: from op-amp design to aerospace applications. *Control Systems, IEEE*, 25(3):44–51. 14

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. 19

Lessin, D., Fussell, D., and Miikkulainen, R. (2013). Open-ended behavioral complexity for evolved virtual creatures. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) 2013*, pages 335–342. ACM. 77

Lessin, D. and Risi, S. (2015). Soft-body muscles for evolved virtual creatures: The next step on a bio-mimetic path to meaningful morphological complexity. In *Proceedings of the European Conference on Artificial Life 2015*, pages 604–611. 55

Lipson, H., Sunspiral, V., Bongard, J., and Cheney, N. (2016). On the difficulty of co-optimizing morphology and control in evolved virtual creatures. *The 2018 Conference on Artificial Life: A Hybrid of the European Conference on Artificial Life (ECAL) and the International Conference on the Synthesis and Simulation of Living Systems (ALIFE)*, pages 226–233. 77

Lukoševičius, M. and Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149. 9, 13

Lundberg, K. (2005). The history of analog computing: introduction to the special section. *Control Systems, IEEE*, 25(3):22–25. 14

Maass, W., Natschläger, T., and Markram, H. (2003). Computational models for generic cortical microcircuits. In Feng, J., editor, *Computational neuroscience: A comprehensive approach*, pages 575–605. Chapman & Hall/CRC, London. 13, 28

Magnenat, S., Waibel, M., and Beyeler, A. (2013). Enki - an open source fast 2d robot simulator (v1.0). `http://home.gna.org/enki/`. stephane.magnenat@epfl.ch, markus.waibel@epfl.ch, antoine.beyeler@epfl.ch. 39

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. 18

McGeer, T. (1990). Passive dynamic walking. *The International Journal of Robotics Research*, 9(2):62–82. 11, 13, 14, 54

Mirletz, B. T., Park, I.-W., Flemons, T. E., Agogino, A. K., Quinn, R. D., and SunSpiral, V. (2014). Design and control of modular spine-like tensegrity structures. In *Proceedings of The 6th World Conference of the International Association for Structural Control and Monitoring (6WCSCM)*. 55

Mirolli, M. (2012). Representations in dynamical embodied agents: Re-analyzing a minimally cognitive model agent. *Cognitive science*, 36(5):870–895. 15

Moore, J. M., McGowan, C. P., and McKinley, P. K. (2015). Evaluating the effect of a flexible spine on the evolution of quadrupedal gaits. In *Proceedings of the European Conference on Artificial Life 2015*, pages 166–173. 55, 72

Müller, V. C. and Hoffmann, M. (2017). What is morphological computation? on how the body contributes to cognition and control. *Artificial Life*, 23(1):1–24. 10, 11, 69, 78

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. 9, 10, 14, 45, 54, 55, 72

Nakajima, K., Hauser, H., Li, T., and Pfeifer, R. (2015). Information processing via physical soft body. *Scientific reports*, 5:10487. 10, 45

Niven, J. E. and Laughlin, S. B. (2008). Energy limitation as a selective pressure on the evolution of sensory systems. *Journal of Experimental Biology*, 211(11):1792–1804. 70

NVIDIA (2016). Physx, *availble at https://developer.nvidia.com/gameworks-physx-overview*. 55

Paul, C. (2004). *Investigation of Morphology and Control in Biped Locomotion*. PhD thesis, Department of Computer Science, University of Zurich, Switzerland. 6, 7, 54

Paul, C. (2006). Morphological computation: A basis for the analysis of morphology and control requirements. *Robotics and Autonomous Systems*, 54(8):619–630. 8, 14

Pfeifer, R. (1996). Building "fungus eaters": Design principles of autonomous agents. In Maes, P., Wilson, S. W., and Mataric, M. J., editors, *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 3–12, Cambridge, MA. MIT Press. 13

Pfeifer, R. and Bongard, J. (2006). *How the body shapes the way we think: a new view of intelligence*. MIT Press, Cambridge, MA. 6, 7, 10, 13, 37, 54

Pfeifer, R. and Iida, F. (2005). Morphological computation: Connecting body, brain and environment. *Japanese Scientific Monthly*, 58(2):48–54. 6, 7, 13, 14, 54

Pfeifer, R., Iida, F., and Gómez, G. (2006). Morphological computation for adaptive behavior and cognition. In *International Congress Series*, volume 1291, pages 22–29. Elsevier. 6, 7, 11

Pfeifer, R., Iida, F., and Lungarella, M. (2014). Cognition from the bottom up: on biological inspiration, body morphology, and soft materials. *Trends in cognitive sciences*, 18(8):404–413. 10, 11, 13, 14

Pfeifer, R. and Scheier, C. (1999). *Understanding intelligence*. MIT Press, Cambridge, MA. 6, 53

Purcell, E. M. (1977). Life at low reynolds number. *American journal of physics*, 45(1):3–11. 66

Rieffel, J., Knox, D., Smith, S., and Trimmer, B. (2014). Growing and evolving soft robots. *Artificial life*, 20(1):143–162. 54, 55

Rieffel, J. A., Valero-Cuevas, F. J., and Lipson, H. (2010). Morphological communication: exploiting coupled dynamics in a complex mechanical structure to achieve locomotion. *Journal of The Royal Society Interface*, 7(45):613–621. 8, 55, 56

Rosenblueth, A. and Wiener, N. (1950). Purposeful and non-purposeful behavior. *Philosophy of Science*, 17(4):318–326. 69

Salumäe, T., Rañó, I., Akanyeti, O., and Kruusmaa, M. (2012). Against the flow: A braitenberg controller for a fish robot. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 4210–4215. IEEE. 70

Scarr, G. (2014). *Biotensegrity, The Structural Basis of Life*. Handspring Publishing, United Kingdom. 8

Schaal, S., Mohajerian, P., and Ijspeert, A. (2007). Dynamics systems vs. optimal control—a unifying view. *Progress in brain research*, 165:425–445. 6

Schramm, L. and Sendhoff, B. (2011). An animat's cell doctrine. In *ECAL 2011: Proceedings of the 11th European Conference on the Synthesis and Simulation of Living Systems*, pages 739–746, Cambridge, MA. MIT press. 45, 51, 55, 72

Sfakiotakis, M. and Tsakiris, D. P. (2006). Simuun: A simulation environment for undulatory locomotion. *International Journal of Modelling and Simulation*, 26(4):350–358. 55, 71, 79

Shim, Y. and Husbands, P. (2012). Chaotic exploration and learning of locomotion behaviors. *Neural computation*, 24(8):2185–2222. 13, 54

Shim, Y. and Husbands, P. (2015). Incremental embodied chaotic exploration of self-organized motor behaviors with proprioceptor adaptation. *Frontiers in Robotics and AI*, 2:7. 54

Shintake, J., Ming, A., and Shimojo, M. (2011). A novel propulsion method of flexible underwater robots. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4735–4740. IEEE. 70

Sims, K. (1994). Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22, New York, NY. ACM. 1, 45, 54, 77

Smith, R. (2016). Open dynamics engine, *available at www.ode.org*. 55

Tamm, S. L. (1984). Mechanical synchronization of ciliary beating within comb plates of ctenophores. *Journal of experimental biology*, 113(1):401–408. 69

Tedrake, R., Zhang, T. W., fai Fong, M., and Seung, H. S. (2004). Actuating a simple 3d passive dynamic walker. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 5, pages 4656–4661 Vol.5. 11

Terzopoulos, D., Tu, X., and Grzeszczuk, R. (1994). Artificial fishes: Autonomous locomotion, perception, behavior, and learning in a simulated physical world. *Artificial Life*, 1(4):327–351. 55, 56

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. 14

Vogel, S. (2003). *Comparative Biomechanics: Life's Physical World*. Princeton University Press, Princeton. 37

Von Uexküll, J. (1957). A stroll through the worlds of animals and men: A picture book of invisible worlds. In Schiller, C. H., editor, *Instinctive Behavior - The development of a modern concept*, pages 5–80. International Universities Press, Inc. 60

Watanabe, W. and Ishiguro, A. (2007). Improvement of learning efficiency by exploiting multiarticular muscles - a case study with a 2d serpentine robot. In *SICE Annual Conference 2007*, pages 2155–2160. 8

Wilson, S. W. (1991). The animat path to ai. In Meyer, J. A. and Wilson, S. W., editors, *From Animals to Animats: Proceedings of the First International Conference on the Simulation of Adaptive Behavior*, pages 15–21, Cambridge, MA. The MIT Press/Bradford Books. 45, 54

Wittmeier, S., Jäntsch, M., Dalamagkidis, K., Rickert, M., Marques, H. G., and Knoll, A. (2011). Caliper: A universal robot simulation framework for tendon-driven robots. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 1063–1068. IEEE. 55

Yamauchi, B. and Beer, R. (1994). Integrating reactive, sequential, and learning behavior using dynamical neural networks. In Cliff, D., Husbands, P., Meyer, J.-A., and Wilson, S. W., editors, *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 382–391, Cambridge, MA. MIT Press. 14

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. 10, 13, 14, 48, 72

Ziegler, M., Iida, F., and Pfeifer, R. (2006). "Cheap" underwater locomotion: Roles of morphological properties and behavioural diversity. *Proceedings of Climbing and Walking Robots*. 70