ORIGINAL PAPER

Biological Cybernetics

Towards a theoretical foundation for morphological computation with compliant bodies

**Helmut Hauser · Auke J. Ijspeert ·
Rudolf M. Füchslin · Rolf Pfeifer · Wolfgang Maass**

Abstract The control of compliant robots is, due to their often nonlinear and complex dynamics, inherently difficult. The vision of morphological computation proposes to view these aspects not only as problems, but rather also as parts of the solution. Non-rigid body parts are not seen anymore as imperfect realizations of rigid body parts, but rather as potential computational resources. The applicability of this vision has already been demonstrated for a variety of complex robot control problems. Nevertheless, a theoretical basis for understanding the capabilities and limitations of morphological computation has been missing so far. We present a model for morphological computation with compliant bodies, where a precise mathematical characterization of the potential computational contribution of a complex physical body is feasible. The theory suggests that complexity and nonlinearity, typically unwanted properties of robots, are desired features in order to provide computational power. We demonstrate that simple generic models of physical bodies, based on mass-spring systems, can be used to implement complex nonlinear operators. By adding a simple readout (which is static and linear) to the morphology, such devices are able to emulate complex mappings of input to output streams in continuous time. Hence, by outsourcing parts of the computation to the physical body, the difficult problem of learning to control a complex body, could be reduced to a simple and perspicuous learning task, which can not get stuck in local minima of an error function.

Keywords Morphological computation · Embodiment ·
Analog computation · Volterra series ·
Nonlinear mass-spring systems

H. Hauser (✉) · R. M. Füchslin · R. Pfeifer
Artificial Intelligence Laboratory, Department of Informatics,
University of Zurich, Andreasstrasse 15, 8050 Zurich, Switzerland
e-mail: hhauser@ifi.uzh.ch

R. M. Füchslin
ZHAW Zurich University of Applied Sciences, Center for Applied
Mathematics and Physics ZAMP, 8401 Winterthur, Switzerland

A. J. Ijspeert
École Polytechnique Fédérale de Lausanne, Biorobotics
Laboratory BIOROB, 1015 Lausanne, Switzerland

W. Maass
Graz University of Technology, Institute for Theoretical Computer
Science, 8010 Graz, Austria

1 Introduction

Most classical robot designs are based on rigid body parts connected by high torque servos and a central controller to coordinate them. This approach follows the view that the physical body is some complex (dynamic) system, which has to be dominated by a cleverly designed central controller. Although this is the standard approach, the resulting robots typically perform poorly compared to their biological role models. They are rather inflexible, exhibit jerky movements and tend to have a high energy consumption, see for example, Collins et al. (2005). On the other hand, the vision of morphological computation proposes a radical different point of view, see Pfeifer and Bongard (2007). Instead of suppressing the complex dynamics introduced by the compliant physical body, which is the reason why classical robots are built of rigid parts, the body could be potentially employed as a computational resource. This suggests that at least a part of the computations, which are needed during interaction, could be outsourced to the physical body itself. Hence, the body is

⚛ Springer

not seen anymore as a device, which is deemed to merely drag the brain around, but rather that it is highly involved in computational tasks. As a result, the remaining learning or control task and its implementation is less complex, than it would be without the aid of the physical body.

The term morphological computation is rather general. It not only includes a broad range of different levels of complexity regarding computation but also embraces a huge variety of different morphologies (e.g., on the molecular level as well on the level of biological organisms).[1] The theoretical framework, which we will present, is not able to cover all these possible types of morphological computation. We will use the term in the context of generic models (based on mass-spring systems) of muscle–skeleton systems of biological systems and the corresponding compliant structures in robots. We address morphological computation in the context where it is possible to outsource relevant parts of the computation to the morphology (i.e., to the compliant physical body). As a consequence, the morphology will allow us to reduce the complex task of emulating nonlinear computation to the much simpler task to adapt some linear parameters for an additional readout. Regarding the type of computation, we consider mathematical models, which can be characterized as complex mappings of input to output streams in continuous time.

There are a lot of cases of biological systems, which suggest that the concept of morphological computation is of value for real-world applications. For a number of examples and a general discussion of morphological computation, we refer to Pfeifer and Bongard (2007). Inspired by that idea, different robots have been designed. A rigorous implementation of this concept are passive walkers. The first of a series was developed by McGeer (1990). Typically, such a robot has no active controller at all. Only its passive physical structure maintains the balance in a robust fashion, while it walks down a slope. Therefore, one could argue that the computation, which is needed in order to balance the robot robustly, is "computed" by the physical body itself. A further development are passive walkers with attached (active) controllers in order to enable the robots to walk even on flat ground, e.g., Wisse and Frankenhuyzen (2003). The used controllers are remarkably simple, since most of the "computational work" is done by the physical body. A clever design does not only simplify the controlling task but also the task to learn to control. For example, Tedrake et al. (2005) showed that the complexity of the task to learn to walk was drastically reduced by the use of a passive walker. Due to the design of the physical

structure of the robot, the system was able to explore online different walking strategies without loosing balance.

Next to the two-legged walking robots, there exist also a number of biologically inspired robots, which mimic a range of species by simultaneously implementing the concept of morphological computation. For example, the simple quadruped robot by Iida and Pfeifer (2006) with a mixture of active and passive joints exhibits a surprisingly robust behavior, although no explicit control feedback is used. Another successful implementation is the artificial fish "Wanda" by Ziegler et al. (2006). It exploits the dynamics between its physical body and its environment. In the physically more complex field of flying has also been demonstrated that morphological computation can play an important role, for example, to stabilize flight, e.g., Wood (2007) and Shim and Husbands (2007).

Another more abstract implementation of the idea of morphological computation are tensegrity robots, see Paul et al. (2006). These robots are built of a special combination of rigid struts and compliant strings. Already simple controllers (found by genetic algorithms) were able to induce locomotion by indirectly exploiting the dynamics of the physical body.

Despite the large body of evidence, which suggests that morphology plays an important role in the successful interaction of complex bodies with their environment, so far there has been no rigorous theoretical basis for this phenomenon. As far as the authors know, there has been only one attempt by Paul (2006). Her line of argumentation, based on real-world and thought experiments, resulted in the heuristic that a physical body with a greater amount of "dynamic coupling" (complexity) has a higher possibility of a reduced control requirement. While her statement is correct, as we see later, it is rather vague. On the other hand, we will provide a precise mathematical model to describe the computational power of physical bodies. This will not only enable us to grasp the capabilities and limitations of morphological computation but also will give us insight of how to construct physical bodies in order to be computationally more versatile than others.

This raises the question: What type of computation is useful for biological systems and, therefore, for biologically inspired robots? Classical computation models, such as Turing machines, simply map a batch of input numbers in an offline computation onto output numbers. However, this type of computation is far from the needs of a robot, which should act in a real environment. It has to integrate continuously information from various continuous input streams (sensory information) and map them onto multiple output streams (motor control). Typically, such streams are mathematically encoded as functions of time. Computations, which map from such continuous input streams to a continuous output stream,

---

[1] It is also closely related to the concept of *embodiment*, which is the dynamic and reciprocal coupling among brain (control), body, and environment as defined in Pfeifer et al. (2007).
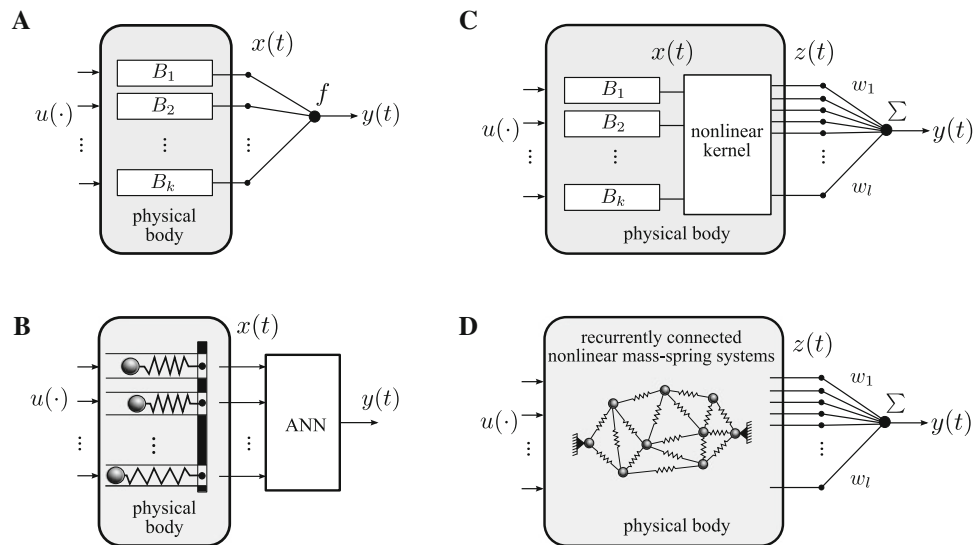
Fig. 1 From abstract theoretical models for morphological computation to real physical bodies (consisting of mass-spring systems). **a** The morphology (represented here by an array of randomly chosen, time-invariant, fading memory filters $B_1, \ldots, B_k$) contributes all temporal integration that is required to approximate a given filter $\mathcal{F}$. The readout f is here some memoryless, continuous function and provides the necessary nonlinear combination. Our theory provides evidence for a surprisingly large computational power of this simple architecture. **b** A possible implementation of (**a**) with a physical body. The filter array is built of an array of linear mass-spring systems and the readout is implemented by a feedforward artificial neural network (ANN). **c** In this architecture, the morphology contributes, in addition to the temporal integration via fading memory filters, generic nonlinear preprocessing in the form of some arbitrary kernel (i.e., nonlinear projection of $x(t)$ into a higher dimensional space). In this case, only a *linear* readout (instead of e.g., an feedforward ANN) has to be added externally. **d** A possible physical realization of (**c**). The array of filters and the kernel are both implemented by a randomly connected network of nonlinear springs and masses. In the resulting computational device, the output weights $[w_{\text{out},1}, \ldots, w_{\text{out},l}]$ are the only parameters, which are adapted in order to approximate a given complex filter $\mathcal{F}$

are referred to operators or filters. We will use here the expression *filter*,[2] denoted by $\mathcal{F}$. In principle, the computation of a filter $\mathcal{F}$ involves two nontrivial computational processes. First temporal integration of information (which is needed if the current output $y(t)$ does not depend only on the actual input $u(t)$ but also on the values $u(s)$ for some time points $s < t$), and, second, the nonlinear combination of such temporally integrated information.

We will provide two theoretical models, each of which is able to represent both computational processes.

The considered models are depicted in Fig. 1a, c. We will demonstrate that both of them can be implemented with the help of generic physical bodies, provided that they are sufficiently complex, i.e., non-rigid and diverse. Figure 1b, d depicts two proposed corresponding real physical implementations of these models with mass-spring systems. Note that physical bodies of biological systems as well of compliant robots can be described by such mass-spring systems. We will provide proofs that such physical realizations tend to represent the two theoret-

ical models and, therefore, emulate their computational powers. Furthermore, we will present a number of simulations to support this view.

For both models, we are able to demonstrate (with simulations) the contribution of the morphological structure to the computation. In the first setup (Fig. 1a, b), the morphological structure contributes only the temporal integration. Therefore, in order to complete the computation, a *nonlinear*, but static readout has to be added. In the second setup (Fig. 1c, d), the morphology provides both necessary computational processes (i.e., temporal integration and nonlinear combination). As a consequence, only a *linear*, static readout is needed. The corresponding linear "weights" can be calculated by some simple, supervised algorithms, such as linear regression (LR), but our setup also offers the potential use of some reward based (as in Legenstein et al. 2010) or even completely unsupervised learning rules, such as Slow Feature Analysis as proposed in Wiskott and Sejnowski (2002). To put it in other words, the learning of complex, nonlinear dynamic filters can be reduced, through the help of the physical body (morphology), to the much simpler task of learning some static, linear weights. This perspective points to a particularly interesting feature of morphological computation, namely that it facilitates the *learning* of complex filters. Usually the learning of such filters requires nonlinear

---

[2] Note that although the term filter is often associated with somewhat trivial signal processing or preprocessing devices one should not fall into the trap of identifying the general term of a filter, as we use it here, with special classes of filters, like, for example, linear filters.

optimization procedures, which often get stuck in local minima of the error function and, which also tend to generalize not too well to new inputs. However, since the morphological computation reduces this learning problem to the learning of some static output weights, it is guaranteed that learning can not get stuck in local minima of the mean squared error (mse) function and has arguably optimal generalization capabilities,[3] see Bartlett and Maass (2003).

In addition, we demonstrate in our simulations that a rather arbitrarily given (or "found") physical body can be employed for such morphological computations, since the parameters of the simulated physical bodies were not optimized for the approximation of a given filter $\mathcal{F}$, but rather randomly chosen from a given probability distribution.[4] This implies that the same physical body can in principle be used for carrying out many morphological computations simultaneously by using a corresponding number of readouts from this physical body. In other words, multitasking of morphological computations is an inherent property of the setups that we describe in this article.

In the next section, we provide the theoretical foundations for morphological computations and prove that our proposed physical implementations with mass-spring systems are valid physical realizations of the theoretical models. In Sects. 3 and 4, we present various simulations to support the results of the theoretical analysis. Finally, we conclude with a discussion.

2 Theoretical foundations

In this section, we present the theoretical foundations for morphological computation. We will show that certain (generic) types of physical bodies (i.e., which consist of mass-spring systems) can be exploited as computational resources. Enhanced only by a static (memoryless) readout they can be used to approximate uniformly any given filter $\mathcal{F}$ (linear or nonlinear) from the class of *time-invariant* filters with *fading memory*. The restriction to time-invariant, fading memory filters is requested by the theory we provide. However, such a restriction is not a drawback at all, since all

physical systems are time-invariant and a lot of practically interesting filters have the property of fading memory.

Preliminary let us clarify the notation we use. We are considering computations, which map from functions (or vector of functions) to functions. We we will refer to them as filters $\mathcal{F}$. The input is denoted by $u : \mathbb{R} \to \mathbb{R}^n$ and the output by y. The argument t of $u(t)$ and $y(t)$ is interpreted as the time point t. The input domain is denoted by U. Therefore, we write for the filter $\mathcal{F} : U \to \mathbb{R}^{\mathbb{R}}$, where $\mathbb{R}^{\mathbb{R}}$ is the class of all functions from $\mathbb{R}$ to $\mathbb{R}$. In order to express that the output $y(t)$ at time t is the result of applying the filter $\mathcal{F}$ to an input u, we write $y(t) = (\mathcal{F}u)(t)$.

Now we are ready to define the desired properties of time invariance and fading memory for the considered filters.

Fading memory is a continuity property of filters. It requires that for any input function $u(\cdot) \in U$, the output $(\mathcal{F}u)(0)$ can be approximated by the outputs $(\mathcal{F}v)(0)$ for any other input function $v(\cdot) \in U$ that approximated $u(\cdot)$ on a sufficiently long time interval $[-T, 0]$ in the past.[5] Thus, in order to approximate $(\mathcal{F}u)(0)$, it is not necessary to know the *precise* value of the input function $u(s)$ for any time s, and it is also not necessary to have knowledge about values of $u(\cdot)$ for more than a finite time interval back into the past.

Time-invariant filters are filters, which can be computed by devices that are input-driven, in the sense that the output does not depend on any absolute internal clock of the computational device. Formally one says, a filter $\mathcal{F}$ is *time-invariant*, if any temporal shift of the input function $u(\cdot)$ by some amount t_0 causes a temporal shift of the output function by the same amount t_0, i.e., $(\mathcal{F}u^{t_0})(t) = (\mathcal{F}u)(t + t_0)$ for all $t, t_0 \in \mathbb{R}$, where u^{t_0} is the function defined by $u^{t_0}(t) := u(t + t_0)$. Note that if the domain U of input functions $u(\cdot)$ is closed under temporal shifts, then a time-invariant filter $\mathcal{F} : U \to \mathbb{R}^{\mathbb{R}}$ is characterized uniquely by the values $y(0) = (\mathcal{F}u)(0)$ of its output functions $y(\cdot)$ at time 0. In other words, in order to characterize a time-invariant filter $\mathcal{F}$, we just have to observe its output values at time 0, while its input varies over all functions $u(\cdot) \in U$.

Another way to characterize nonlinear, time-invariant filters with fading memory is to describe them with Volterra series.[6] A Volterra series is a finite or infinite sum (with $d = 0, 1, \ldots$) of terms of the form

$$\int_0^\infty \ldots \int_0^\infty h_d(\tau_1, \ldots, \tau_d) \cdot u(t - \tau_1) \cdot \ldots \cdot u(t - \tau_d) d\tau_1 \ldots d\tau_d,$$

---

[3] Results from statistical learning theory, see Vapnik (1998), imply that the test error of any classifier from a hypothesis class $\mathcal{H}$ can be bounded from above by the error on the set of training examples (drawn from the same distribution D as the test set), plus a term that grows with the VC-dimension of $\mathcal{H}$. This upper bound holds for any distribution D, hence, also if there are correlations among different coordinates of examples (therefore, this upper bound can also be applied to readouts from a reservoir). The hypothesis class $\mathcal{H}$ of linear classifiers over examples of dimension n has VC-dimension $n + 1$ (see Bartlett and Maass 2003), which is the smallest VC-dimension of any nontrivial class $\mathcal{H}$ of classifiers (that allows that classifiers take all n coordinates into account).

[4] Note that the optimization of the readout is a convex optimization problem, however, "optimizing" the body (i.e., its parameters) is not.

[5] Formally one defines: A filter $\mathcal{F} : U \to \mathbb{R}^{\mathbb{R}}$ has fading memory, if for every $u \in U$ and every $\varepsilon > 0$ there exist $\delta > 0$ and $T > 0$ so that $|(\mathcal{F}v)(0) - (\mathcal{F}u)(0)| < \varepsilon$ for all $v \in U$ with $\|u(t) - v(t)\| < \delta$ for all $t \in [-T, 0]$.

[6] In fact, under some mild conditions on the domain U of input streams, the class of time-invariant, fading memory filters coincides with the class of filters, which can be characterized by Volterra series.

where some integral kernel h_d is applied to products of degree d of the input stream $u(\cdot)$ at various time points $t - \tau_i$ back in the past.

Note that the Volterra series presentation is rather general and, therefore, is able to describe a number of interesting filters. For example, it is possible to express a simple integration of information over time, i.e., memory. However, more interestingly in the context of robotics, a Volterra series is also able to represent any continuous, nonlinear dynamical system with a single exponentially stable equilibrium point, for a proof please refer to Boyd (1985). Since our proof will take the Volterra series presentation as a basis, as a consequence, our morphological computation devices, which will introduce here, are in principle able to emulate the same class of complex filters.

In order to show that such complex filters $\mathcal{F}$ can be approximated with the help of certain types of physical bodies (which consist of mass-spring systems), we use a theoretical result from Boyd and Chua (1985). This result builds on the Stone–Weierstrass approximation theorem and it implies that arbitrary time-invariant filters with fading memory can be uniformly approximated by computational devices, which consist of two stages:

- an array or *filter bank* of finitely many "basis filters" $B_1, \ldots, B_k$ in parallel that all receive the same input function $u : \mathbb{R} \to \mathbb{R}^n$, and which are all assumed to be time-invariant with fading memory
- a memoryless (i.e., *static*) readout function $f : \mathbb{R}^k \to \mathbb{R}$ that maps the vector of outputs $x(t) = \langle (B_1 u)(t) \ldots, (B_k u)(t) \rangle$ of the first stage at time t onto some output $y(t)$.

Figure 1a reflects this setup. A remarkable fact, which provides the basis for our theoretical analysis of morphological computation, is that the basis filters $B_1, \ldots, B_k$ of the filter bank are not required to be of a particular form. Rather, they can be chosen from any pool of time-invariant, fading memory filters,[7] which satisfies the following pointwise separation property.

Definition A class **B** of basis filters has the pointwise separation property, if there exists for any two input functions $u(\cdot)$, $v(\cdot)$ with $u(s) \neq v(s)$ for some $s \leq t$ a basis filter $B \in \mathbf{B}$ with $(Bu)(t) \neq (Bv)(t)$.

This pointwise separation property is satisfied by simple, explicitly defined classes **B**, such as the class of tapped delay lines. However, it tends to be satisfied also by classes **B** of "found" physical realizations of linear and nonlinear filters.

We will show that linear mass-spring systems are one type of such physically realizable filters, which form a class **B**, which has the pointwise separation property. An interesting fact is that, although no conditions are imposed on particular filters of **B**, a substantial diversity among the filters in **B** is required. A remarkable consequence is that a physical implementation of such a filter bank (in form of a morphological structure) has to exhibit this substantial diversity. While classical approaches to control robots try to avoid such complexity, or at least try to reduce it, our theoretical model of morphological computation demands it and, therefore, provides potentially an explanation of the complexity of biological systems.[8]

Based on the definition of the pointwise separation property and the theorem and proof provided by Boyd and Chua (1985), we can state the following theorem:

Theorem *Any time-invariant filter $\mathcal{F}$ with fading memory that maps some n-dimensional input stream $u \in U$ onto an output stream y can be approximated with any desired degree of precision by the simple computational model shown in Fig. 1a,*

1. *if there is a rich enough pool **B** of basis filters (time-invariant, with fading memory), from which the basis filters $B_1, \ldots, B_k$ in the filter bank can be chosen (**B** needs to have the pointwise separation property) and*
2. *if there is a rich enough pool **R** from which the readout functions f can be chosen (**R** needs to have the universal approximation property, i.e., any continuous function on a compact domain can be uniformly approximated by functions from **R**).*

For a detailed proof, we refer to Theorem 1 in Maass et al. (2002) and Theorem 3.1 in Maass and Sontag (2000) and their corresponding proofs. They applied this theory to artificial and spiking neural networks. However, we are going to use this mathematical framework in the context of morphological computation. Hence, we will employ models of compliant bodies of biological systems and real robots instead of neural networks. In order to apply the presented theory to a morphological computation setup, we have to decide on how to implement the basis filters and the readout function as depicted in Fig. 1a. One possibility is to use real physical linear mass-spring systems to build the filter bank and a feedforward artificial neural network (ANN) as readout function. In order to show that this choice is consistent with the previously stated theorem, we have to demonstrate that linear mass-spring systems are time-invariant, have fading memory and that a pool of such systems has the pointwise separation property. Regarding the

---

[7] Note that the class of nonlinear filters $\mathcal{F}$, which we want to approximate by our computational device, is much richer than the class of filters, which can be used to build the filter bank.

[8] Note that Paul (2006) came to a similar conclusion.

readout, we have to demonstrate that a pool of ANNs has the universal approximation property. However, the latter one has already been proven by Hornik et al. (1989), demonstrating that already feedforward networks with one hidden layer exhibit this property. Note that in a biological system the nonlinear readout might be implemented by a biological neural network.

This leaves us with the task to prove the validity of using linear mass-spring systems to build the filter bank. A single linear mass-spring system can be described by following equations:

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = -\frac{k}{m}x_1 - \frac{d}{m}x_2 + \frac{1}{m}u \qquad (1)$$
$$y = x_1,$$

where x_1 is the displacement relative to the resting length l_0 of the spring, x_2 the rate of change of x_1 (velocity $\dot{x}_1$), $k \in \mathbb{R}^+$ the linear spring constant, $d \in \mathbb{R}^+$ the linear damping constant, m the mass of the endpoint, and u the sum of all external forces acting on the mass. First, it can be easily seen that the dynamic system of Eq. 1 is time-invariant. Second, we have to show that the system has the property of fading memory. Since it is finite-dimensional and linear, it is sufficient to demonstrate that it is exponentially stable, see Sect. 5.1 in Boyd and Chua (1985). The eigenvalues of the system are $s_{1,2} = -d/2m \pm \sqrt{(d/2m)^2 - (k/m)}$. Since in real physical realizations of such systems $k, m \in \mathbb{R}_{>0}$, the real part $-d/2m$ is negative for any values of k and m. Hence, the system is exponentially stable and, therefore, has the property of fading memory. Third, the pointwise separation property of a pool of similar systems was discussed in Sect. 5.2 of Boyd and Chua (1985), where it was shown that this property holds for a special class of systems, i.e., Wiener's Laguerre systems. It can be shown that not only this special subset has the pointwise separation property, but any class of finite-dimensional, linear dynamic systems, see Boyd and Chua (1985), to which the system of Eq. 1 also belongs to. Hence, real physical linear mass-spring systems can be used as basis filters $B_1, \ldots, B_k$ in the setup with feedforward [9] mass-spring systems as depicted in Fig. 1b.

Of course there exist a number of other possible implementations. A closely related morphology in a biological system is the structure of the wings of a bird. A number of diverse feathers receive the same input (i.e., air pressure) and mechanoreceptors measure the distortions. This could represent a biological implementation of the filter bank of our proposed theoretical model. Remarkably, the resulting morphological computation has already been considered in Shim and Husbands (2007). They used nonlinear angular springs to simulate the distortions of the feathers and combined it with

simulated mechanoreceptors and a neural network (i.e., nonlinear readout). The network weights were found by genetic algorithms. While their design was inspired by the biological system itself, we provide here a theoretical model, which is able to explain their results.

So far, we have only considered a setup with a clear separation of the temporal integration (implemented by a filter bank of linear mass-spring systems) and the nonlinear combination (implemented by an ANN). However, one could also consider to merge both stages into one morphological structure. As a consequence, the physical body would be then not only responsible for the temporal integration (as in the filter bank setup) but also for the nonlinear combination (see Fig. 1c). In this context, one could choose for **R** a pool of functions consisting of a fixed nonlinear *kernel*. The notion of a kernel[10] that we use here is closely related to the notion of a kernel for Support Vector Machines in machine learning as in Vapnik (1998). However, whereas a kernel for a Support Vector Machine is a virtual mathematical concept, we are considering here concrete physical implementations of a kernel. As a consequence, such a kernel can only satisfy the kernel property for a fixed finite range. However, sufficiently large and randomly connected analog circuit of sufficiently many and diverse nonlinear components tend to map a large class of pairwise different inputs onto linear independent outputs. Therefore, a particularly tempting option for morphological computation is to let both, the filter bank and the kernel, be realized by a single physical body. We will demonstrate with the help of simulations that random, recurrently connected networks of *nonlinear* springs and masses tend to have this "kernel property." In other words, such a physical body tends to carry out temporal integration and nonlinear combination at once. Note that in contrast to the setup with feedforward mass-spring systems, where the readout was an ANN, in this case only an additional *linear* readout is required. Hence, learning to approximate a given nonlinear (time-invariant, fading memory) filter $\mathcal{F}$ is reduced to the simpler task of learning some weights, providing a number of advantages as already discussed in the introduction. Figure 1c depicts this idea of combing both computational processes in one physical body. Figure 1d depicts the corresponding proposed physical implementation as a random, recurrent network of nonlinear springs and masses. In the context of biological

---

[9] As opposed to the *recurrent* networks as sketched in Fig. 1c,d.

[10] A kernel (in the sense of machine learning) is a nonlinear projection Q of k input variables $u_1, \ldots, u_k$ into some high-dimensional space. For example, all products $u_i \cdot u_j$ could be added as further components to the k-dimensional input vector $\langle u_1, \ldots, u_k \rangle$. Such nonlinear projection Q boosts the power of any *linear* readout applied to $Q(\mathbf{u})$. For example, in the case where $Q(\mathbf{u})$ contains all products $u_i \cdot u_j$, a subsequent linear readout has the same expressive capability as quadratic readouts f applied to the original input variables $u_1, \ldots, u_k$. More abstractly, Q should map all inputs $\mathbf{u}$ that need to be separated by a readout onto a set of linearly independent vectors $Q(\mathbf{u})$.

systems, such networks can serve as a model to describe the complex and compliant properties of their muscle–skeleton systems. Moreover, such generic structures can be employed to model compliant body parts of a robot.

Note that it is not possible to conclude from our proposed theory anything about the performances of the proposed morphological computation devices (neither for the feedforward structure of Fig. 1a nor for the recurrent network of Fig. 1c). The theory only states that sufficiently large morphological computation systems of the proposed types will provide satisfactory approximation capabilities, as long as the morphology is dynamic and sufficiently diverse. However, for a given filter $\mathcal{F}$, the theory does not provide bounds for the required size of the approximating system. Neither is it possible to conclude directly[11] from the theory, which mass-spring systems (i.e., which physical properties) are needed for a well performing morphological computation device, except that a diversity of network components is desirable. Hence, we employ mass-spring systems with a diversity of masses and spring parameters in order to construct our generic models for compliant bodies.

We present a number of simulations of the proposed physical implementations (Fig. 1b, d) applied to real-world computational tasks (which are of interest for robotics) and demonstrate that already relatively small generic structures can be used to emulate complex, nonlinear filters $\mathcal{F}$.

3 Morphological computation with feedforward mass-spring systems

In this section, we present simulations of the proposed physical realization of the morphological computation setup with feedforward mass-spring systems (Fig. 1b). The simulations consisted of an array of parallel linear mass-spring systems (each of them described by Eq. 1). All static, but nonlinear readouts were implemented as feedforward neural networks, each of them with one hidden layer of sigmoidal neurons and one linear gate as output. In the simulation, we used a generic morphological structure, i.e., the values, which defined the properties of the involved mass-spring systems (i.e., spring constants k and damping constants d) were drawn randomly from a defined range. The simulations were implemented in Matlab and simulated at a time step of 1 ms.

We demonstrate that our proposed morphological computation device with feedforward mass-spring systems is in principle able to emulate a Volterra series operator. In order to have a clear, but nontrivial example, we chose a Volterra series consisting of a quadratic term with a Gaussian kernel.
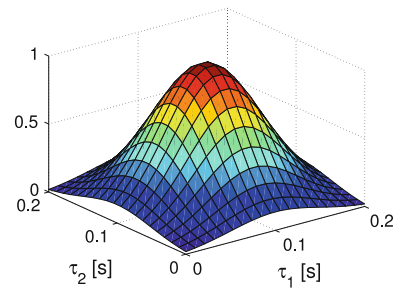
Fig. 2 Quadratic kernel $h_2(\tau_1, \tau_2)$ used to define a Volterra series operator $\mathcal{V}$, which should be approximated by our morphological structure with feedforward mass-spring systems in combination with a nonlinear readout (i.e., as depicted in Fig. 1b)

The chosen Volterra series operator $\mathcal{V}$ is of the form

$$
\begin{aligned}
y(t) &= \mathcal{V}u(t) \\
&= \int\limits_{\tau_1, \tau_2 \in \mathbb{R}_0^+} \int h_2(\tau_1, \tau_2)u(t - \tau_1)u(t - \tau_2)d\tau_1 d\tau_2, \quad (2)
\end{aligned}
$$

where $u(t)$ is the input and h_2 is a Gaussian kernel with $\mu_1 = \mu_2 = 0.1$, and $\sigma_1 = \sigma_2 = 0.05$ (in seconds), i.e., $h_2(\tau_1, \tau_2) = \exp\left((\tau_1 - \mu_1)^2/2\sigma_1^2 + (\tau_2 - \mu_2)^2/2\sigma_2^2\right)$, which is defined for $\tau_1, \tau_2 \in [0, 0.2]$ s. A plot of the kernel can be seen in Fig. 2.

For the simulations, we used a discretized version of the kernel with a discretization step of 1 ms. Note that this computation is not only a simple memorization of past event, but it is more complex. Any computational model, which should approximate this Volterra series operator $\mathcal{V}$, must provide temporal integration (the delays τ_1 and τ_2) and nonlinearity (the quadratic term $u(t - \tau_1)u(t - \tau_2)$).

For the input, we chose a product of three sinusoidal functions with different frequencies: $u(t) = \sin(2\pi f_1 t) \cdot \sin(2\pi f_2 t) \cdot \sin(2\pi f_3 t)$ with $f_1 = 2.11$, $f_2 = 3.73$, and $f_3 = 4.33$ Hz. The resulting signal has a period of 100 s. After some transitional setting time to get rid of the initial conditions of the mass-spring systems, we used 30 s for learning, the subsequent 10 s for validation, and the consecutive 10 s for testing. The first 5 s of the testing data can be seen in Fig. 3a. The result from applying the given Volterra series operator $\mathcal{V}$ to this input signal $u(t)$ is used as target output for the computational device. The first 5 s of the target output for the testing data can be seen in Fig. 3c, red line.

The input signal $u(t)$ was applied to ten linear mass-spring systems (filter bank). They all had different, random spring and damping constants. The values were randomly drawn from a log-uniform distribution from the interval [0.1, 150]. The responses of all linear mass-spring systems to the input can be seen in Fig. 3b. They served as inputs to the ANN, which consisted of ten hidden sigmoidal nodes and one linear gate as output. The weights of the ANN were adapted via BFGS quasi-Newton algorithm. The learning process was
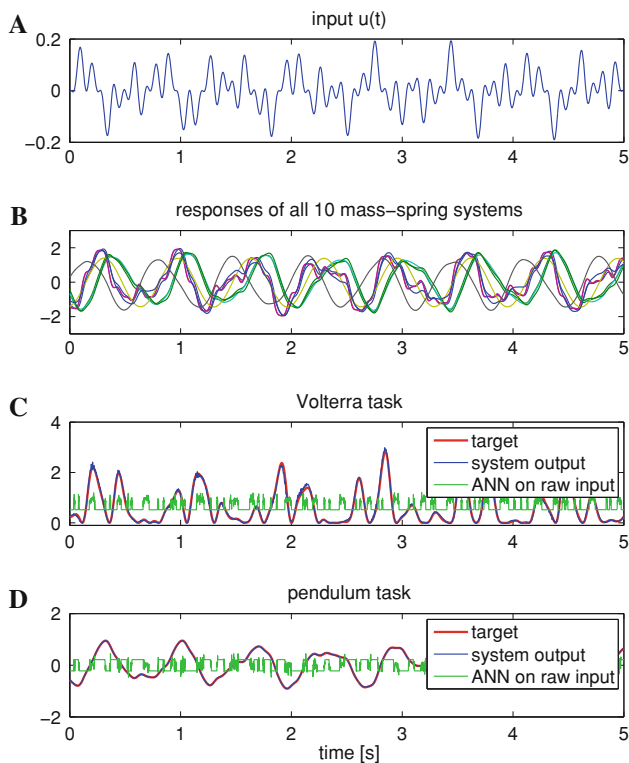
Fig. 3 Applying a feedforward morphological computation device to approximate the Volterra series operator $\mathcal{V}$ (defined by Eq. 2) and the pendulum (Eq. 3) simultaneously with one morphological structure (i.e., multitasking). **a** The used input signal $u(t)$, which consisted of a product of three different sinusoidal functions ($f_1 = 2.11$, $f_2 = 3.73$, and $f_3 = 4.33$ Hz). **b** The responses of all ten mass-spring systems to this input (for a better readability the outputs were normalized to zero mean and a standard deviation of one). **c** The performance of the proposed morphological computation device for the Volterra task. The *red line* is the target (applying the Volterra series operator to the input, i.e., $\mathcal{V}u(t)$) and the *blue line* shows the output of the morphological computation device. The *green line* shows the performance of the device, when no morphological structure was available, i.e., only the nonlinear readout of the ANN was applied to the raw input data. Clearly this approach fails, since the ANN is only a static readout and is not able to represent the necessary temporal integration, which was contributed in the previous case by the morphological structure. **d** The pendulum task: the *red line* is the target, the *blue line* the output of the morphological computation device and the *green line*, when no morphological structure was available

terminated, when the error of the validation data started to increase. For more details, please refer to the supplementary material. Figure 3c shows the performance after learning. The red line is the target signal, i.e., $\mathcal{V}u(t)$, and the blue line is the output of our morphological computational device. The achieved mse was 6.83×10^{-3} on the testing data.

In order to demonstrate the contribution of the morphological structure to the computation, we compared the results to the case when no physical body (no array of mass-spring systems) was available and only the nonlinear readout (i.e., ANN) on the raw input signal remained. In order to have the

same number of weights, the ANN was resized accordingly. The results can be seen in Fig. 3c. The green line is the output of the plain ANN after learning. One can see clearly that this approach failed to emulate the given Volterra series operator. The reason is that the ANN is only a static readout and is not able to represent the necessary temporal integration, which was contributed in the previous case by the morphological structure. As already argued in the introduction, the setup offers the possibility of multitasking, i.e., the same fixed[12] morphological structure can potentially be used for a number of different tasks. Note that the ability of multitasking is a beneficial feature, since the morphological structures of real robots (and biological systems) are to a high degree fixed.[13] In order to demonstrate multitasking, we used the same morphological structure (same filter bank) and the same input of the previous task and applied it to a new task by simply adding a new readout (i.e., an additional ANN).

For the additional task, we chose from an interesting subclass of nonlinear filters, which can be described by Volterra series, namely the class of nonlinear dynamical system with fading memory.[14] An example of such a dynamical system is the damped pendulum, which can be described by the following equations (taken from Khalil 2002):

$$\dot{\alpha} = \omega$$
$$\dot{\omega} = -\frac{g}{l}\sin(\alpha) - \frac{\mu}{m}\omega + \frac{1}{ml^2}A\tau \qquad (3)$$
$$y = \alpha,$$

where α is the angle, ω the angular velocity, $g = 9.81 \ m/s^2$ the gravitational acceleration, l the length, m the mass of the bob, and μ the friction coefficient. The constant A is a proportional factor, which was set to $A = 40$ in order to drive the system into the nonlinear domain of the state space. For the same reason, we set in the simulations $l = 0.5$, $m = 0.1$ and $d = 1$. The input to the system was the torque τ and the output was the angle α. In order to obtain suitable targets, we simulated the system of Eq. 3 at a time step of 1 ms with Matlab's internal ordinary differential equation solver. The input $u(t)$, now interpreted as torque $\tau(t)$, was the same as in the previous task (Fig. 3a). The red line in Fig. 3d shows the corresponding output (i.e., target).

Since we used the same morphological structure (the same filter bank array) and the same input $u(t)$, consequently, the responses of the mass-spring systems were the same as before

---

[12] Note that "fixed" is used in the sense of fixing the parameters, which describe the physical models of the springs. The mass-spring systems themselves have to be dynamic.

[13] However, for the biological case exists experimental evidence, that the stiffness can change in order to adapt to different environments (e.g., Ferris et al. 1988).

[14] Boyd and Chua (1985) have pointed out that fading memory for (time-invariant) dynamical systems is related to the *unique steady-state property*. For a discussion, we refer to Boyd (1985).

(Fig. 3b). Based on these responses as inputs, an ANN with ten hidden sigmoidal neurons and one linear output gate was trained (with the BFGS quasi-Newton algorithm) to approximate the desired targets. The performance can be seen in Fig. 3d. The resulting mse was 1.29×10^{-4}. Thus, the fixed generic morphological structure in conjunction with two different readouts was able to represent the two different nonlinear filters very well.

Again, in order to make the contribution of the morphological structure to the computation explicit, we compared the results to the case when no physical body (array of mass-spring systems) was available and only the ANN remained. The performance can be seen in Fig. 3d, where the green line represents the output of the ANN. As before, the ANN applied to the raw input stream failed to represent the desired nonlinear filter (i.e., the pendulum equations).

As previously argued, the morphological structure has to be diverse in order to be computationally powerful. In order to show that this is true, we set the properties of all the mass-spring systems in the filter bank to the same values (k and d were the same). The best resulting mse for the Volterra task in this case was 0.960, which was more than 140 times higher than with the previously used heterogeneous filterbank.

4 Morphological computation with recurrent networks of nonlinear springs and masses

In the previous simulations, we used the approach with a strict feedforward structure (Fig. 1a). It implemented a spatial separation between a linear but dynamic part (implemented as an array of linear mass-spring systems), which provided temporal integration, and the nonlinear, static readout (implemented as a feedforward ANN), which provided the nonlinearity. However, as we have already argued in Sect. 2, there could exist physical realizations which have the property to combine both computational aspects in a single body. We will demonstrate in the following simulations, that random, recurrent networks of nonlinear springs and masses tend to be such physical realizations. A particular interesting property of this setup is that in contrast to the setup with feedforward mass-spring systems, where a nonlinear readout (e.g., ANN) was needed, in this case only a simple *linear* readout has to be added in order to *complete* the morphological computation (compare Fig. 1b, d). In addition, such networks can serve as generic models to describe the complex and nonlinear dynamics of the compliant bodies of robots and biological systems (i.e., muscle–skeleton system).

We continue with a description of the implementation of the simulation of such networks followed by a number of example tasks.

4.1 Implementation of recurrent networks of nonlinear springs and masses

We considered an implementation of random, recurrent networks of nonlinear springs and masses, to which we refer as *mass-spring networks* or simple as *networks*. In the next sections, we describe how we constructed such networks, how we simulated them, and how we implemented the learning process for the linear readout.

4.1.1 Constructing mass-spring networks

The construction of the mass-spring networks was based on the following two principles. First, the final network should be realizable as a real physical system, and second, it should be generic, i.e., not constructed for any specific task.

A chosen number of N nodes (mass points) were randomly positioned (uniformly distributed) within a defined range of a two-dimensional plane. Subsequently, we connected these mass points by nonlinear springs. In order to find reasonable, non-crossing spring connections, we calculated a Delaunay triangulation on this set of points, resulting in L non-crossing spring connections. A schematic example of such a mass-spring network can be seen in Fig. 4. Every single nonlinear spring of such a network can be described by the following nonlinear dynamic system:

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = -p(x_1) - q(x_2) + u, \tag{4}$$

where $x_1 = l - l_0$ is the difference between the actual length l and the resting length l_0, $x_2 \in \mathbb{R}$ is the rate of change of x_1 (velocity $\dot{x}_1$), and u the sum of all external forces acting on it. At the beginning of the simulation, we assumed the mass-spring network to be at rest (i.e., all springs were at their point of equilibrium $\mathbf{x_i} = [0, 0]^T$ for $i = 1, \ldots, N$ and therefore all masses were at rest). In order to accomplish this we set as per definition the resting lengths l_0 of all nonlinear springs to the distances (at the start of the simulation) between the mass nodes they connected, hence $l_0 := l(t = 0)$. The functions p and q were nonlinear and, in order to have a stable and physically reasonable system, had to be monotonically increasing and fulfill $p(0) = 0$ and $q(0) = 0$.[15] Typically nonlinear springs are modeled by 3rd order polynomials, e.g., as described in Palm (1999). Therefore, we implemented the nonlinear functions as $p(x_1) = k_3 x_1^3 + k_1 x_1$ and $q(x_2) = d_3 x_2^3 + d_1 x_2$, where $k_1, d_1 \in \mathbb{R}_{>0}$ and $k_3, d_3 \in \mathbb{R}^+$ defined the properties of the spring. In order to get a rich kernel, as argued in Sect. 2, the springs should be diverse.

---

[15] A proof for that is based on the Lyapunov function $V(\mathbf{x}) = \int_0^{x_1} p(\zeta)d\zeta + \frac{1}{2}x_2^2$, its derivative $\dot{V}(\mathbf{x}) = -x_2 q(x_2)$ and the use of a corollary of La Salle's Theorem (see Theorem 4.4 and Corollary 4.2 in Khalil (2002)).
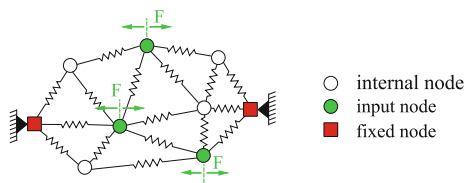
Fig. 4 Schematic example of a generic mass-spring network. The nodes (masses) are connected by nonlinear springs. The *red* nodes are fixed in order to hold the network in place. The *green* nodes are randomly chosen inputs nodes, which receive the input in form of horizontal forces scaled by randomly initiated weights

Fig. 5 Implementation of input, linear readout, and simulation of forces of the mass-spring networks. **a** The input is applied to an input node as a horizontal force F_x proportional to the input signal u (scaled by a randomly initialized weight w_{in} for this input node). **b** The readout from the network is the weighted sum of all L spring lengths $y(t) = \sum_{i=1}^{L} w_{\text{out},i} l_i(t)$. In general, the input as well as the output can be multidimensional. **c** All the spring forces act along their spring axis. The resulting force F_{sum} is the sum of all forces acting on the node and is found by the summation of the force vectors

Hence, the parameters describing the spring properties (i.e., k_1, k_3, d_1, and d_3) were randomly drawn from a defined range, assigned to the connections and subsequently fixed. The left most and the right most mass nodes were fixed in order to keep the network in place (red squares in Fig. 4).

A certain percentage of all nodes were randomly chosen to be input nodes (green nodes in Fig. 4). During simulation, they received a linearly scaled version of the current input in form of a horizontal force. Before the simulation started, the input scaling factors (weights $\mathbf{w}_{in} = [w_{in,1}, w_{in,2}, \ldots]^T$) had been randomly drawn from a certain range and had been fixed subsequently.

The linear readout of the network was defined as the weighted sum of all actual spring lengths $y(t) := \sum_{i=1}^{L} w_{\text{out},i} l_i(t)$. The output weights ($\mathbf{w}_{\text{out}} = [w_{\text{out},1}, w_{\text{out},2}, \ldots, w_{\text{out},L}]^T$), in contrast the rest of the network parameters, were adapted in the learning process.

4.1.2 Simulating mass-spring networks

We simulated every single mass points (of a total number of N) at a time step of 1 ms by the following equations:

$$m\ddot{p}_x = F_x + w_{in}u \tag{5}$$

$$m\ddot{p}_y = F_y, \tag{6}$$

where $\ddot{p}_x$ and $\ddot{p}_y$ were the accelerations of the mass point relative to a global reference frame split up into its two spatial dimensions, F_x and F_y were the forces acting on the mass in the corresponding spacial dimensions, and $w_{in}u$ was the weighted input. Note that the input was defined as a horizontal force (see Fig. 5a) and if the mass point was no input node $w_{in}u := 0$. We chose forces as input form, since for a real compliant robot any interaction with its environment results in forces acting on it.[16] For the sake of simplicity,[17] all

masses were set to $m = 1$. The forces F_x and F_y resulted from the nonlinear springs, which were connected to this mass point. The forces they applied to the mass point depended on the states of the nonlinear springs, i.e., x_1 and x_2 in Eq. 4. The value of x_1 was calculated by the actual length $l(t)$ (Euclidean distance between the two masses which the spring connected) and the resting length l_0. The velocity x_2 was approximated by $(x_1(t) - x_1(t - \Delta t))/\Delta t$ with a time step of $\Delta t = 1$ ms. The resulting forces were calculated by the nonlinear functions $p(x_1)$ and $q(x_2)$. This procedure was repeated for all springs connected to the mass. We assumed that these forces acted along their corresponding spring axes. Finally, all spring forces acting on the regarding mass node were summed up vectorially (see Fig. 5c). Subsequently, the resulting force F_{sum} was split up into its two spatial dimensions and added as forces F_x and F_y to Eqs. 5 and 6. If the mass point was an input node, the current input $u(t)$ was added in form of a scaled horizontal force (see Eq. 5 and Fig. 5a). The new position and velocity of the mass were found by integrating Eqs. 5 and 6 numerically with the fourth-order Runge–Kutta method. The same procedure was repeated for all masses. At the end of the simulation step, the current output was calculated by a linear combination of the actual lengths of all springs, i.e., $y(t) = \sum_{i=1}^{L} w_{\text{out},i} l_i(t)$ (see Fig. 5b).

4.1.3 Learning the linear readout of mass-spring networks.

The structure of the mass-spring networks, as well as the parameters, which defined the physical behavior, were randomly initialized and subsequently fixed. Only the linear readout was adapted during the learning process, i.e., the weights $\mathbf{w}_{\text{out}} = [w_{\text{out},1}, w_{\text{out},2}, \ldots, w_{\text{out},L}]^T$ were adapted. For learning, we considered a networks of N nodes connected by L springs. During the simulation, we collected the current lengths of every single spring $l_i(t)$ for $i = 1, \ldots, L$

---

[16] We chose the forces to be horizontal for the sake of simplicity, however, the proposed setup works for other force directions too.

[17] Note that the masses are only linear scaling factors and, since the properties of the springs were randomly drawn, could be set to 1 for all masses without the loss of generality. Nevertheless, in a real biological
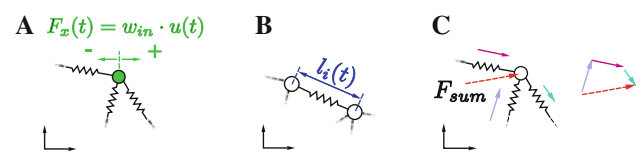
body (or robot) a diversity of masses is natural and it contributes further diversity.
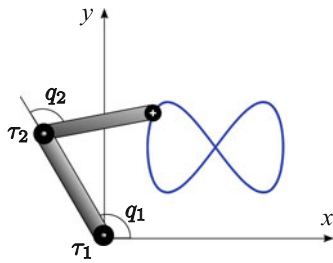
Fig. 6 Setup for the robot arm task. The *blue line* is the desired trajectory for the end-effector

Fig. 7 Generic mass-spring network used for the robot arm task and subsequently for the multitasking task in Sect. 4.3. The *red* nodes are globally fixed and the *green* nodes are the randomly chosen input nodes. The network consisted of 30 masses and 78 nonlinear springs

at every time step $t = 1, \ldots, M$ in a $L \times M$ matrix $\mathbf{L}$. We dismissed data from an initial period of time (washout time) to get rid of initial transients. The target signal was also collected over time in a matrix $\mathbf{T}$. Finally, the optimal values for the output weights were calculated by $\mathbf{w}_{\text{out}}^* = \mathbf{L}^\dagger \mathbf{T}$, with $\mathbf{L}^\dagger$ being the (Moore–Penrose) pseudoinverse, since in general $\mathbf{L}$ was not a square matrix. Note that the same procedure can be applied in the case of multiple inputs and/or multiple outputs.

4.2 Representing inverse dynamics by a recurrent mass-spring network

As a first task, we will demonstrate that a generic mass-spring network can be used to learn the complex mapping of the end-effector trajectory of a robot arm in Cartesian space to its corresponding torques for a given trajectory (i.e., it is able to represent the inverse dynamics for a given trajectory and velocity). Note that we do not try to learn the full model of the inverse dynamics, but rather we demonstrate that mass-spring systems can be used to learn directly a given mapping. This is somewhat less complex than emulating a nonlinear filter, nevertheless, it is a relevant task for a number of possible applications for our proposed setup.

We used a full dynamic model of a two link robot arm from Slotine and Li (1991), which was assumed to move in a horizontal plane. Hence, the gravitational forces were ignored. We refer to the supplementary material for further details on the robot model. Figure 6 shows the setup of the task.

The end-effector of the robot arm had to move along the blue trajectory. The corresponding trajectories in Cartesian space, i.e., x and y positions, are plotted in Fig. 8a. The corresponding targets torques, which allowed the robot arm to move along these trajectories, can be seen in Fig. 8c (red lines). These torques were found by the following process: We chose an arbitrary starting posture. Based on the x- and y-trajectories (which defined the Fig. 8 trajectory in Cartesian space) and the Jacobian of the robot arm we calculated the corresponding trajectories of the joint angles. Subsequently, the corresponding torques were found by the
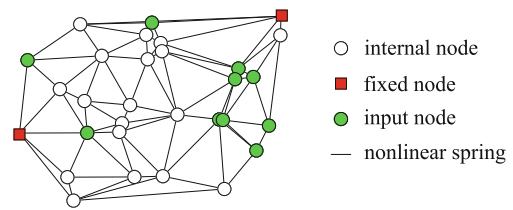
use of PD-controllers [18] in order to follow those joint angle trajectories.

We constructed a generic mass-spring network based on the previously described process (Sect. 4.1.1). The parameters of the springs (i.e., k_1, k_3, d_1, and d_3 as defined in Sect. 4.1.1) were randomly drawn from the range [1, 100] (log-uniform distribution) for the values k_1 and d_1, and from [100, 200] (uniform distribution) for the values k_3 and d_3. We chose randomly 20% of all nodes to be input nodes for the first input (i.e., input signal x) and also 20% of all nodes for the second input (i.e., y). For more details, please refer to the supplementary material. One of the obtained mass-spring networks can be seen in Fig. 7. It consisted of 30 masses and 78 nonlinear springs.

As described in Sect. 4, the randomly chosen input nodes (green nodes) received a scaled horizontal force proportional to the input. The scaling weights $\mathbf{w}_{\text{in}}$ were randomly (uniform distribution) drawn from $[-1, +1]$. The mass-spring network responded to this inputs by changing the mass positions and the spring lengths. Figure 8b shows ten typical spring length trajectories (out of all 78). For a better readability, the trajectories in this plot were normalized to zero mean and a standard deviation of one. Based on the targets, all 78 spring length trajectories and the previously described learning process we calculated the optimal output weights. Note that these weights were static, i.e., did not provide temporal integration, and that the resulting readout was linear, i.e., it did not provide any nonlinearity. Figure 8c shows the performance after learning (using the network of Fig. 7). The red lines are the target torques and the blue lines are the outputs of the morphological computation device (solid blue for τ_1 and dashed blue for τ_2). We can see that the setup was able to represent the dynamic mapping from the Cartesian space to the robot arm torques.

In order to demonstrate the contribution of the morphological structure to the computation, we compared the results to the case when no physical body (i.e., no mass-spring network) was available and only the linear readout remained.

---

[18] The used P and D values were empirically found to have a reasonable performance.

In order to do so, we applied LR on the raw input signals. Therefore, we defined the output at time t by $\tau_1^{LR}(t) = w_1 x(t) + w_2 y(t) + w_{bias}$, where x and y were the inputs (as in Fig. 8a) and $\mathbf{w}^{LR} = [w_1, w_2, w_{bias}]^T$ were some static weights, which were found by standard LR. Accordingly, we calculated the three corresponding weights for the second output τ_2. Figure 8d shows the performance of this approach. The red lines are the targets and the green lines are the outputs. The approach failed because it was no able to represent the necessary temporal integration and nonlinear combination. In the previous case (with the physical body), the morphological structure provided both of these computational aspects.[19]

The network of Fig. 7 was chosen based on the fact that it was the best performing network out of 400 networks constructed with the same probability distribution, i.e., the same construction parameters, which defined the ranges for the random values used for the construction process. More specifically, these construction parameters were the defined ranges, from which the spring parameters were drawn from, the percentage of all nodes, which received an input, the range for the input weights $\mathbf{w}_{in}$ and the size of the area in which we randomly placed all mass points. This raises the question whether it is easy to find such a set of parameters, which defines a pool of well performing networks. Note that, for example, the range of possible values for the spring parameters k_1 and d_1 went over two decades ([1, 100]—see description above). This points to the fact that no tedious parameter tuning was necessary. In order to demonstrate that the used (rather broad ranged) construction parameters defined a whole set of well performing networks, and the presented network was not just a statistical outlier, we constructed 400 random networks using exactly these parameters. Subsequently, we sorted the networks according to their performances (i.e., by their averaged mean squared error over both outputs; denoted here by mse). The results are presented in Fig. 8f. We can see that even the worst performing network had still a mse smaller than 10^{-3}. Out of the 400 mass-spring networks we chose the best network (blue line), the worst (black dotted line), and the median network (green line).[20] The table in Fig. 8 lists the mse of them. Figure 8c shows the performance of the best network. Figure 8e shows the performances of the worst network (black dotted line) and the median network (green line). Similar results can be obtained for other tasks and construction parameters. This suggests that in general no tedious parameter search has to be done in order to find probability distributions to define a successful pool of networks. This means that the physical body does not
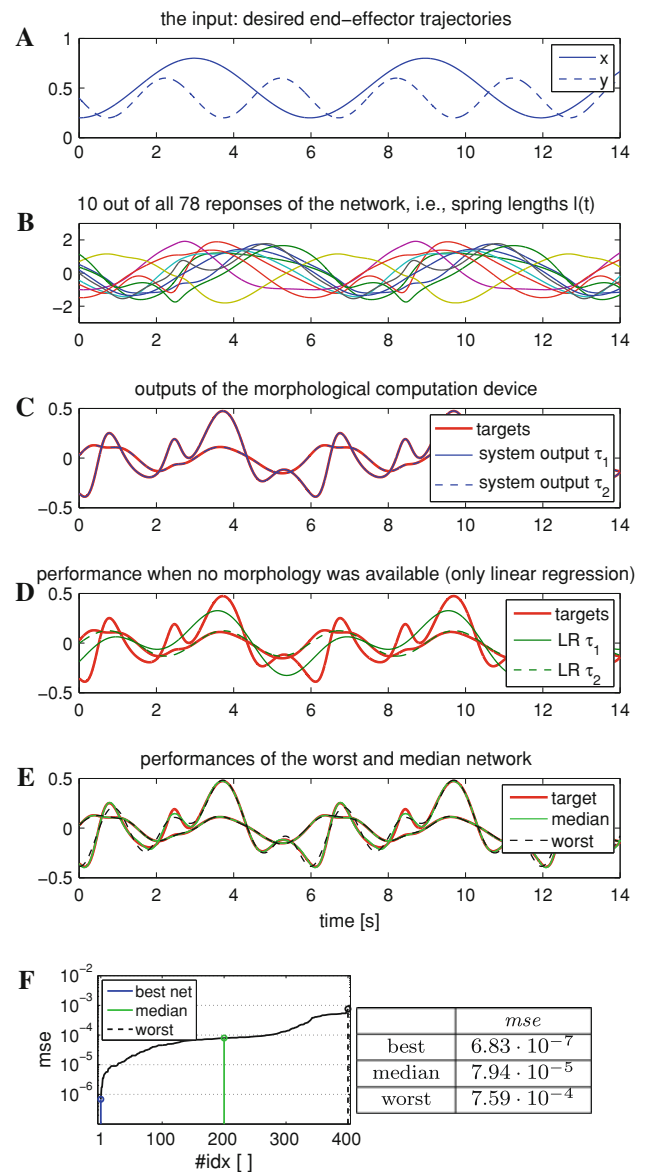

Fig. 8 Representation of the inverse dynamics of a robot arm with the help of morphological computation. **a** The desired end-effector trajectory split up in its two Cartesian coordinates x and y (i.e., inputs). **b** Ten typical responses (out of all 78) of the mass-spring network to this input. For a better readability, each signal was normalized to zero mean and a standard deviation of one. **c** The performance of the morphological computation device. The *red lines* are the target torque trajectories and the *blue lines* are the outputs of the computational device. **d** The performance when no morphological structure was available, i.e., only a LR on the actual values of the inputs remained. This approach failed to represent the dynamic and nonlinear mapping. **e, f** Based on the same construction parameters, we randomly generated 400 networks and sorted them by their mean squared error (*mse*) over its two outputs. The table shows the performances of the best, the worst, and the median network. The best network was used for the plot of (**c**). The performances of the worst (*black dotted line*) and the median network (*green*) are presented in (**f**)

---

[19] Note that in the setup used in the Sect. 3 the physical body only provided the temporal integration.

[20] By median, we mean that half of all networks had a better and the other half had an equal or worse performance.

have to be tuned for a specific task in order to be a valid computational resource, as long as it is sufficiently complex and diverse. Therefore, the same morphological structure could be potentially used for a number of different tasks simultaneously (i.e., multitasking).

We argued in the Sect. 2 that a diversity, i.e., different physical parameters, is an important property of a computationally powerful physical body. In order to demonstrate the validity of this assumption, we simulated the same network structure (Fig. 7), but now with all spring having the same physical parameters. We set all k_1 values to the average value of the previously used network. Accordingly, we set the values for k_3, d_1, and d_3. The averaged *mse* over both outputs was 5.2×10^{-3}, which was 7,600 times higher than the best randomly found network and still about seven times higher than the worst one.

4.3 Multitasking property of a mass-spring network

In the previous section, we demonstrated that mass-spring networks can be employed to represent a direct mapping. However, the theory suggests that the setup is more powerful and that it even can be employed to emulate complex, nonlinear filters. In this context, we present here three different tasks (i.e., filters to emulate) and, additionally, we demonstrate that mass-spring networks have the desired property of multitasking. By multitasking, we refer to the ability to carry out various computations simultaneously—in our context, employing the same physical body for different computational tasks at the same time. Note that in contrast to the multitasking in the setup with feedforward mass-spring systems (Sect. 3), where we used different ANNs as readouts, in the case of mass-spring networks only a corresponding number of *linear* readouts is sufficient. For the following simulations, we used therefore one generic network, one input, and three different linear readouts to emulate three different nonlinear filters.

For the first target filter, we chose the previously defined Volterra series operator $\mathcal{V}$ (Eq. 2). The second task was to emulate following second-order nonlinear dynamic system

$$y[k + 1] = 0.4y[k] + 0.4y[k]y[k - 1] + 0.6u^3[k] + 0.1 \tag{7}$$

where $u[k]$ was the input and $y[k]$ the output at time step k. The third task was to emulate following nonlinear tenth-order system

$$y[k + 1] = 0.3y[k] + 0.05y[k]\left(\sum_{i=0}^{9} y[k - i]\right) \tag{8}$$
$$+ 1.5u[k - 9]u[k] + 0.1 \, .$$

Again $u[k]$ was the input and $y[k]$ was the output at time step k. The systems 7 and 8 were both taken from Atiya and

Parlos (2000), where they were used in order to demonstrate the performance of a new learning algorithm for recurrent networks.

Note that nonlinear systems of the type of Eq. 8 are typically hard to emulate for recurrent networks due to their long-term time dependencies, see Hochreiter and Schmidhuber (1997). Note also that our proposed morphological computation device is an analog device, which is able to map continuous input streams onto continuous output streams. However, in the simulation of this analog device, we were restricted to discrete time. The simulation time step and the time step of Eqs. 7 and 8 were the same. A real physical (analog) implementation of the morphological computation device would emulate the underlying continuous dynamic systems, which correspond to the discrete Equations 7 and 8 and, which minimize the errors at the discretization time steps.

We used the same network as in the previous robot arm task (Fig. 7). All previously chosen input nodes (in the robot task assigned for two inputs) were now defined to receive the single input $u[k]$. As input, we employed the same signal as previously for the experiment in Sect. 3, where we used the morphological computation device with feedforward mass-spring systems. It was a product of three sinusoidal functions and it is shown again in Fig. 9a. For learning, we used the first 95 s of the signal.[21] The first 50 s were defined as *washout period* and thrown away (see Sect. 4.1.3). Thus, we had 45 s for learning. The subsequent 5 s were used for testing. Figure 9b shows ten typical trajectory (out of all 78) of the spring lengths as responses of the mass-spring network to this input in the testing phase. The output weights for the linear readouts were found as previously described. Figure 9c shows the performance of our morphological computation device for the Volterra task. The red line is the target and the blue line is the output of the morphological computation device. The mass-spring network with an additional linear readout is able to emulate the nonlinear filter defined by the Volterra series operator $\mathcal{V}$. Note that, unlike to the previous Volterra task of Subsect. 3 (with a filter bank), here the physical body (i.e., mass-spring network) provided not only the temporal integration but also the nonlinearity. Hence, in order to learn to emulate the given nonlinear filter $\mathcal{V}$, due to the use of the nonlinear and dynamic morphological structure (as a computational resource), we only had to calculate a simple LR.

In order to show the explicit contribution of the morphological structure to the computation, we compared the results with the case when no physical body (no mass-spring network) was available and only LR on the raw input data remained. We used a LR with two weights, w_1 for the actual input $u(t)$ and w_2 to learn a bias. Hence, the resulting output at time step t was $y^{LR}(t) = w_1 u(t) + w_2$. The performance

---

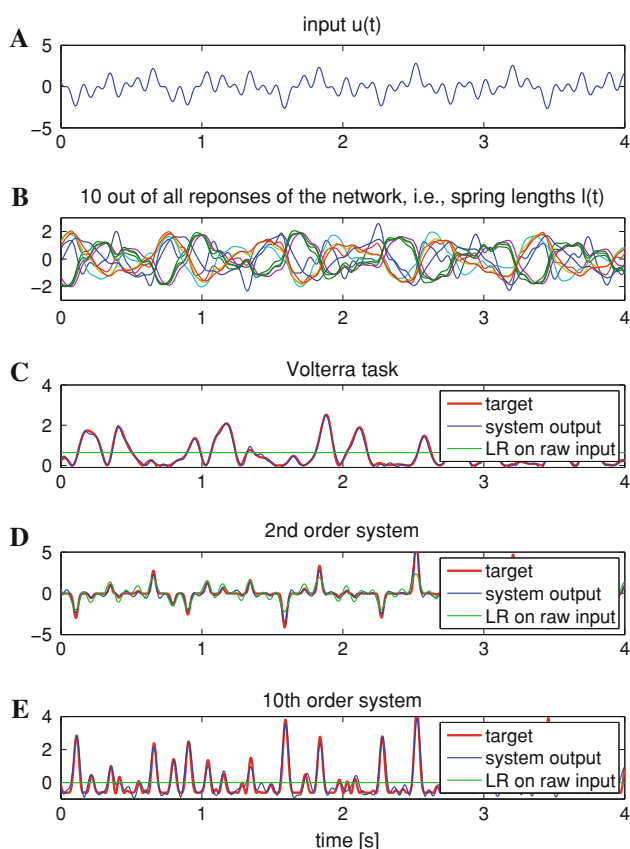[21] Note that the period of the input signal was 100 s.

Fig. 9 Simultaneous morphological computation of the three nonlinear filters with one generic recurrent mass-spring network (i.e., multitasking). **a** The input $u(t)$, which consisted of a product of three sinusoidal functions. **b** The trajectories of ten typical (out of 78) individual spring lengths $l(t)$ as responses to this input. **c** The performance for the Volterra task. The *red line* is the target function and the *blue line* is the output of the morphological computation device. The *green line* depicts the outputs of the device, when no morphological structure was available, i.e., only the linear readout was applied to the raw input data. Note that the result is simply a scaled version of the input with some offset. **d** Performance of emulating system 7. **e** Performance for the filter defined by the system 8

can be seen in Fig. 7c. The output $y^{\mathrm{LR}}(t)$ is depicted by the green line, which is simply a scaled version of the input (with a very small amplitude) with an additional offset. Not surprisingly, pure LR on the raw input stream failed to represent the nonlinear filter $\mathcal{V}$, since all the required temporal integration and nonlinearity was contributed before by the physical body (mass-spring network).

Figure 9d, e shows the performances of the morphological computation device in order to emulate the nonlinear systems 7 and 8 using the same morphological structure (mass-spring network of Fig. 7). Again, the red lines are the targets and the blue lines the outputs of the device. The green lines depict the results, when no morphological structure was available and only pure LR was applied to the input stream. One can see,

that also in these cases, the LR, which was static and linear, failed to represent the necessary dynamics and nonlinearity.

In summary, we can see that one single mass-spring network (i.e., one physical body) can be employed to emulate a number of different nonlinear filters by simple adding a corresponding number of linear and static readouts.

5 Discussion

We introduced two theoretical models, which provide a potential explanation for the computational power of compliant bodies. We applied a theoretical model for computation, which allowed us to demonstrate, how body parts, modeled by multiple mass-spring systems, can be employed to emulate arbitrary, time-invariant, nonlinear filters with fading memory. Since the underlying theoretical framework is not able to provide us with a precise guidance for constructing a well performing morphology for a given computational task, we demonstrated the applicability of the approach by simulating a number of generic morphological structures. These simulations also allowed us to indicate qualitatively the contribution of the morphology to the computational task.

The proposed setups are formed by a dynamic morphological structure (i.e., the physical body with fixed parameters) and a static readout (which can be adapted). As we have shown, the readout can be even linear if the morphological structure is sufficiently *rich*. Remarkably, such simple devices are in principle able to emulate any nonlinear, time-invariant filter with fading memory by adapting a simple, linear readout. Hence, the complex task of learning to emulate such complex filters can, due to the help of the morphological structure (i.e., due to morphological computation), be reduced to the task of finding suitable weights for the linear readout. This suggests that physical bodies are potentially able to boost the expressive power of attached linear learning systems (e.g., the brain or the controller of the robot). Note that the possibility to restrict the readouts to linear ones enables highly efficient learning, because the number of training examples required by a linear learning device is minimal according to general results from learning theory (because of its minimal VC-dimension, see Vapnik 1998). Furthermore, it guaranties that the optimization process does not get stuck in a local minimum and it has optimal generalization capabilities.

The proposed theory suggests that morphological structures, in order to be computationally powerful (in the context of our proposed setups), should be diverse in their parameters and that they should exhibit high dimensionality. Remarkably, both aspects are typical properties of compliant, biological body parts. However, in classical robot design, these attributes are suppressed (by high torque servos and rigid body parts) in order to have a more tractable model and an

easier controllable robot. Our results point to the fact that the consideration of these dynamic features are essential in order to be able to outsource computational tasks to the morphological structure and, therefore, simplify the control of the robot. This perspective suggests that the development of novel high-dimensional readouts from artificial limbs (e.g., acceleration sensors at many locations inside the robot) is a possible way to exploit the morphological structure. The morphologies of biological systems might even be more suitable for this task, since they provide naturally a high number of internal sensory signals and a variety of interconnected dynamic structures. More specifically, the muscle–skeleton system consists of a high number of different parts (i.e., bones, muscles, tendons, etc.), which have a variety of different physical properties. In addition, the biological system senses the state of its body by numerous sensors located all over the body (i.e, somatosensory system). As our theory suggests, both parts are desired for morphological computation.

Another interesting aspect of the approach is that real physical bodies provide the necessary nonlinearities and the temporal integration for *free*. The physical structure simple reacts on its *inputs*. Actually, it is not even necessary to have real physical interpretations of all the available internal signals in order to exploit them for morphological computation. Furthermore, the bodies of real biological systems are not simply computational devices, but they fulfill real functions. For example, they provide animals (and robots) with the capability to locomote and to interact with their environments. Therefore, a next step will be to apply the proposed theory to morphological structures of real robots. This would also involve the step to move from our presented abstract networks, which were be chosen to demonstrate the applicability of our presented theory, to more realistic simulations including the simulation of the interaction between a robot and its environment. In this context, one would have to investigated the impact of real-world conditions on the performance of the proposed setups. For example, typical cases for such real-word conditions are the partial loss of the state of the morphology and/or noisy readouts.

Another remarkably property of the proposed morphological computation devices is their multitasking capability. One morphological structure is able to provide the necessary signals for approximating several different nonlinear filters—only a corresponding number of readouts has to be added. While this multitasking ability is obviously beneficial, since the physical bodies of biological system as well as of robots are to high degree fixed, one would also assume that the computational tasks for the physical body are limited to a set of particular filters. This suggests that physical bodies or different parts of the body could be optimized regarding to their computational tasks. Note that this optimization, however, would not be a convex problem anymore. Nevertheless, the combination of our presented morphological computa-

tion setups with nonlinear optimization schemes could lead to interesting new types of compliant robot parts. The resulting structures would be inhomogeneous and asymmetric as opposed to the examples presented here. This points to the need for a new type of *computational* material science and *computational* robotics, where the geometrical and statistical properties of the fine structure of different materials are analyzed (and optimized) with regard to their suitability to support through morphological computation the computations of a particular range of filters, e.g., filters that are needed to control a robot for a particular range of tasks.

Obviously, these considerations will also open new perspectives for our understanding of the shape and structural properties of the body of biological organisms and, consequently, will lead to new types of biologically inspired robots.

References

Atiya AF, Parlos AG (2000) New results on recurrent network training: unifying the algorithms and accelerating convergence. IEEE Trans Neural Networks 11(3):697–709

Bartlett PL, Maass W (2003) Vapnik-Chervonenkis dimension of neural nets. In: Arbib MA (ed) The handbook of brain theory and neural networks, 2nd edn. MIT Press, Cambridge, pp 1188–1192

Boyd S (1985) Volterra series: engineering fundamentals. PhD thesis, UC Berkeley

Boyd S, Chua L (1985) Fading memory and the problem of approximating nonlinear operators with volterra series. IEEE Trans Circuits Syst 32(11):1150–1161

Collins S, Ruina A, Tedrake R, Wisse M (2005) Efficient bipedal robots based on passive-dynamic walkers. Science 307:1082–1085

Ferris DP, Louie M, Farley CT (1998) Running in the real world: adjusting leg stiffness for different surfaces. Proc Biol Sci 265(1400):989–994

Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9:1735–1780. ISSN 0899-7667

Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural Networks 2(5):359–366

Iida F, Pfeifer R (2006) Sensing through body dynamics. Robot Auton Syst 54(8):631–640

Khalil HK (2002) Nonlinear Systems, 3rd edn, Prentice Hall, Upper Saddle River

Legenstein R, Chase SA, Schwartz AB, Maass W (2010) Functional network reorganization in motor cortex can be explained by reward-modulated Hebbian learning. In: Proceedings of NIPS 2009: advances in neural information processing systems, vol 22. MIT Press, pp 1105–1113.

Maass W, Sontag ED (2000) Neural systems as nonlinear filters.. Neural Comput 12(8):1743–1772

Maass W, Natschlaeger T, Markram H (2002) Real-time computing without stable states: A new framework for neural computation based on perturbations. Neural Comput 14(11):2531–2560

McGeer T (1990) Passive dynamic walking. Int J Robot Res 9(2):62–82 ISSN 0278-3649

Palm WJ III (1999) Modeling, analysis, and control of dynamic systems, 2nd edn. Wiley, New York. ISBN 0-471-07370-9

Paul C (2006) Morphological computation: a basis for the analysis of morphology and control requirements. Robot Auton Syst 54(8):619–630

Paul C, Valero-Cuevas FJ, Lipson H (2006) Design and control of tensegrity robots for locomotion. IEEE Trans Robot 22(5):944–957. ISSN 1552-3098. doi:10.1109/TRO.2006.878980

Pfeifer R, Bongard JC (2007) How the body shapes the way we think. MIT Press, Cambridge. ISBN 0262162393

Pfeifer R, Lungarella M, Iida F (2007) Self-organization, embodiment, and biologically inspired robotics. Science 318:1088–1093

Shim Y, Husbands P (2007) Feathered flyer: integrating morphological computation and sensory reflexes into a physically simulated flapping-wing robot for robust flight manoeuvre. In: Almeida e Costa F et al (eds) ECAL. Springer, Berlin, pp 756–765

Slotine J-J E, Li J-J W (1991) Applied nonlinear control, 1st edn. Prentice Hall, Englewood Cliffs

Tedrake R, Zhang TW, Seung HS (2005) Learning to walk in 20 minutes. In: Proceedings of the fourteenth yale workshop on adaptive and learning systems. Yale University, New Haven, CT, 2005

Vapnik VN (1998) Statistical learning theory. Wiley, New York

Wiskott L, Sejnowski TJ (2002) Slow feature analysis: unsupervised learning of invariances. Neural Comput 14(4):715–770. ISSN 0899-7667

Wisse M, Van Frankenhuyzen J (2003) Design and construction of MIKE; a 2D autonomous biped based on passive dynamic walking. In: Proceedings of international symposium of adaptive motion and animals and machines (AMAM03)

Wood RJ (2007) Design, fabrication, and analysis of a 3DOF, 3cm flapping-wing MAV. IEEE/RSJ international conference on intelligent robots and systems, 2007. IROS 2007, 29 Oct–Nov 2 2007, pp 1576–1581. doi:10.1109/IROS.2007.4399495

Ziegler M, Iida F, Pfeifer R (2006) "Cheap" underwater locomotion: roles of morphological properties and behavioural diversity. In: CLAWAR