Biol Cybern
DOI 10.1007/s00422-012-0516-4

ORIGINAL PAPER

Biological Cybernetics

The role of feedback in morphological computation with compliant bodies

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

Received: 6 February 2012 / Accepted: 10 August 2012

Abstract The generation of robust periodic movements of complex nonlinear robotic systems is inherently difficult, especially, if parts of the robots are compliant. It has previously been proposed that complex nonlinear features of a robot, similarly as in biological organisms, might possibly facilitate its control. This bold hypothesis, commonly referred to as morphological computation, has recently received some theoretical support by Hauser et al. (Biol Cybern 105:355–370, doi:10.1007/s00422-012-0471-0, 2012). We show in this article that this theoretical support can be extended to cover not only the case of fading memory responses to external signals, but also the essential case of autonomous generation of adaptive periodic patterns, as, e.g., needed for locomotion. The theory predicts that feedback into the morphological computing system is necessary and sufficient for such tasks, for which a fading memory is insufficient. We demonstrate the viability of this theoretical analysis through computer simulations of complex nonlinear mass–spring systems that are trained to generate a large diversity of periodic movements by adapting the weights of a simple linear feedback device. Hence, the results of this article substantially enlarge the theoretically tractable application domain of morphological computation in robotics, and also provide new paradigms for understanding control principles of biological organisms.

Keywords Morphological computation · Nonlinear system · Limit cycles · compliant robots

Electronic supplementary material The online version of this article (doi:10.1007/s00422-012-0516-4) contains supplementary material, which is available to authorized users.

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

In classic robot design rigid body parts and high torque servos are used in order to suppress unwanted dynamics. While this approach provides the advantage of dealing with simpler physical models, there are also numerous disadvantages, for example, the high energy consumption compared to biological systems, see Collins et al. (2005), and unnatural movements. A recently emerging subfield of biologically inspired robotics suggests a different approach. It proposes to employ compliant, rather than rigid, body parts, and to include the properties of these compliant bodies, i.e., complexity and nonlinear dynamics, into the design process. As a consequence, a compliant body is not seen as an imperfect realization of a rigid body anymore, but, as demonstrated, e.g., in Hauser et al. (2012) and in this paper, as a potential computational resource.

For a number of successful implementations of this idea in robots as well in biological systems we refer to Pfeifer and Bongard (2007). These cases demonstrate that the dynamics of the morphology of a robot (or biological organism) can contribute, e.g., to the control of balance, to reduce energy consumption, to offer safer interaction with the environment,

Published online: 06 September 2012

to increase robustness, and even to facilitate the adaptation to new environmental conditions. Based on that and the observation that all these tasks include some form of computation, the term *morphological computation* was coined to describe this phenomenon. Note that morphological computation 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). Morphological computation includes a broad range of different levels of complexity regarding the type of computation (i.e., linear, nonlinear, including memory, dynamic, etc.), but also embraces a huge variety of different morphologies (from the molecular level to the large-scale properties of biological organisms).

For example, the term morphological computation is used in the context of self-assembly (see, e.g., Whitesides and Grzybowski (2002) and the "Tribolons" by Miyashita et al. (2011), and membrane computing in order to sort particles (Shaw et al. 2007). Other examples for morphological computation, which can be observed in nature, are the spatial arrangement of receptor cells in the retina, which provides information about the location of the sensory stimulation, or the non-homogeneous arrangement of the ommatidia in insect eyes (more dense towards the front than on the side) in order to compensates for motion parallax, see Franceschini et al. (1992).

While this concept is very broad and covers a wide variety of different possibilities to employ morphology for computation, in this paper we consider only morphologies in the form of generic models of muscle–skeleton systems (based on mass–spring systems) of biological systems and the corresponding compliant structures in robots.

In this context, an example of a rigorous implementation of the concept of morphological computation are passive walkers. The first of a series was developed by McGeer (1990). The passive physical structure maintains the balance of the robot in a robust fashion and, therefore, one could argue that the computation, which is needed in order to balance the robot robustly, is "computed" by the physical body itself. Collins et al. (2005), Wisse and Van Frankenhuyzen (2003), and Wisse and Linde (2007) have shown that through proper exploitation of passive dynamics, actuated stable walking can be also achieved (even on level ground) in remarkably simple ways, because the "computation" is largely performed by the body, so to speak. A clever design does not only simplify the control 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 building on top of a functioning passive dynamic walker.

Next to bipedal 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 Puppy by Iida and Pfeifer (2006) with a mixture of active and passive joints, the artificial fish "Wanda" by Ziegler et al. (2006), or, in the context of the physically more complex field of flying, winged robots by Wood (2007) and Shim and Husbands (2007).

Another more abstract implementation of this concept are tensegrity robots, see Paul et al. (2006) and Rieffel et al. (2008). They 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.

As these examples suggest, locomotion seems to be an especially fruitful field of application for morphological computation. This is not utterly surprising, since compliant parts tend to oscillate and locomotion is typically based on some sort of repetitive patterns.

Despite these success stories of implementations, so far there exists very little research on underlying theoretical principles. As far as the authors know one of the first attempts to formulate a theoretical foundation was made by Paul (2006). Her line of argumentation, based on experimental 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 general. In Hauser et al. (2012) we introduced a new theoretical model for morphological computation We showed how generic models of compliant bodies (i.e., random networks of nonlinear springs and masses) can be employed for computational tasks. More specifically, we demonstrated that in principle any nonlinear, time-invariant operator with fading memory could be emulated by such models of compliant bodies by simply adding a linear, static readout. While this class of computation is very rich and includes a lot of relevant computations, it still has its limitations, namely, the property of fading memory. In particular, for repetitive patterns (as needed for locomotion) a persistent memory is needed. The same is true for computations that involve switching between different persistent internal states, e.g., depending on some sensory input (often referred to as "working memory" in neuroscience).

Therefore, we extend the previous approach by employing a new mathematical framework, which includes feedback loops. This allows us to employ a compliant physical body, which naturally has the property of a fading memory, to emulate computations, which can even include persistent memories, like limit cycles or analog state switching.

A successful implementation to produce such patterns for locomotion with echo state networks was presented by Wyffels and Schrauwen (2009). However, the standard approach is to use networks of coupled oscillators. Due to their rather abstract implementation they can be employed for all kinds of locomotion. For example, Righetti and

Ijspeert (2008) applied them to various quadrupeds, Ijspeert et al. (2007) implemented such a network in order to produce different movements—swimming and walking—for a salamander robot, and Taga (1998) applied the approach to a humanoid robot with a musculo–skeletal system. In general, these implementations have the advantages that they exhibit robustness and that they are generic and can thus be tailored for a specific task. However, a disadvantage is that the parameters involved are either hand-tuned or found by time intensive nonlinear optimization schemes, e.g., genetic algorithms.

By contrast, in this paper, we will propose a setup where the adaptable parameters are linear and static and can be found with simple linear regression. Naturally, this is much faster than any nonlinear optimization scheme and is guaranteed not to get stuck in a local minimum. In addition, we demonstrate that our setup also exhibits the desired robustness and generic applicability.

While locomotion is a particularly interesting application for morphological computation, our proposed setup is more general. The underlying theoretical model is based on rigorous mathematics and gives a clear insight on how physical bodies can be employed for computational tasks. More specifically, we will demonstrate that nonlinear mass–spring systems, which are typically used to describe the dynamics of compliant biological and robotic body parts, can be enhanced by a nonlinear, static feedback and a nonlinear, static readout to emulate in principle any conceivable computation on some analogue input stream. Moreover, we will show, that if the dynamics of the compliant body is sufficiently complex, already a linear feedback and a linear readout is sufficient, since the needed nonlinearities can be "outsourced" to the physical body.[1] A remarkable conclusion of our theory is that high dimensionality and nonlinearity, both properties usually undesired due to the difficulties to control them, are desired attributes for computationally powerful physical bodies. We support our theoretical results by various simulations of random, recurrent networks of nonlinear springs and masses as generic physical bodies.

In the next section, we provide the theoretical framework for morphological computation with feedback, and demonstrate how a physical body can be employed to carry out complex, analog computations. In Sect. 3 we analyze practical implications, and show in Sect. 4 how this theoretical model can be implemented with a real physical body by applying it to generic models of muscle–skeleton systems (based on mass–spring systems) of biological systems and the corresponding compliant structures in robots. In Sect. 5 we provide details on the physical simulations of these models. In order to support our theory we present results of a number of simulations in Sect. 6. In Sect. 7, we conclude with a discussion and a future outlook. In the appendix we provide the mathematical proofs for our theory.

[1] Note that our approach is closely related to the concept of *reservoir computing*, see Maass et al. (2002), Lukoševičius and Jaeger (2009), and Schrauwen et al. (2007). In our approach the compliant, physical body is employed as a natural reservoir.

2 Theoretical foundations

We present a theoretical framework for morphological computation, which is based on a result by Maass et al. (2007). They proved that a certain class of nonlinear dynamical systems (which can have the property of fading memory) gain computational power to emulate arbitrary nonlinear systems (which can have persistent memory), by adding simply a suitable static (memoryless) feedback and a suitable static (memoryless) readout function. Maass et al. (2007) applied their theoretical framework to recurrent networks of different models of neurons and demonstrated that such generic networks gained computational power by adding appropriate feedbacks and readouts. Remarkably, the original dynamic system, i.e., the network of neurons, remained unchanged. Only the static feedback drove the system in order to emulate, in conjunction with a static readout, a given nonlinear target system.

Another remarkable fact, which will provide the basis for our theory, is that the fixed dynamical system is not required to have a particular form. The only requirement is that it belongs to the class S_n of feedback linearizable systems. A prerequisite for a feedback linearizable system is that it can be described in the very general form

$$\mathbf{x}'(t) = f(\mathbf{x}(t)) + g(\mathbf{x}(t)) \cdot v(t), \tag{1}$$

where $\mathbf{x} = [x_1, \ldots, x_n]^T$ is the state vector, $(.)'$ is the derivative in time, $v(t)$ the input, and $f : \mathbb{R}^n \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}^n$ some sufficiently smooth, but otherwise arbitrary nonlinear vector functions. Given a system in the standard form of Eq. 1 one can analyze, with well established tools of nonlinear control theory, if the system is feedback linearizable or not and, therefore, can be employed for computation. For a basic description of this process we refer to Appendix B, for a detailed discussion we refer to Isidori (2001). We will reserve the letter $\mathcal{C}$ to denote feedback linearizable systems of the form of Eq. 1, i.e., $\mathcal{C} \in S_n$.

A useful property of such systems is, as the name already suggests, that they can be transformed by a suitable feedback into a linear system. Actually, this is a standard tool in nonlinear control in order to obtain a linear dynamic system from a nonlinear system, which is then naturally much easier to control. The resulting linear system $\mathcal{L}$ corresponding to the feedback linearizable system $\mathcal{C}$ can be written as

$$\mathcal{L}: \quad \mathbf{x}' = \mathbf{A}_n \mathbf{x}(t) + \mathbf{b}_n v(t), \tag{2}$$

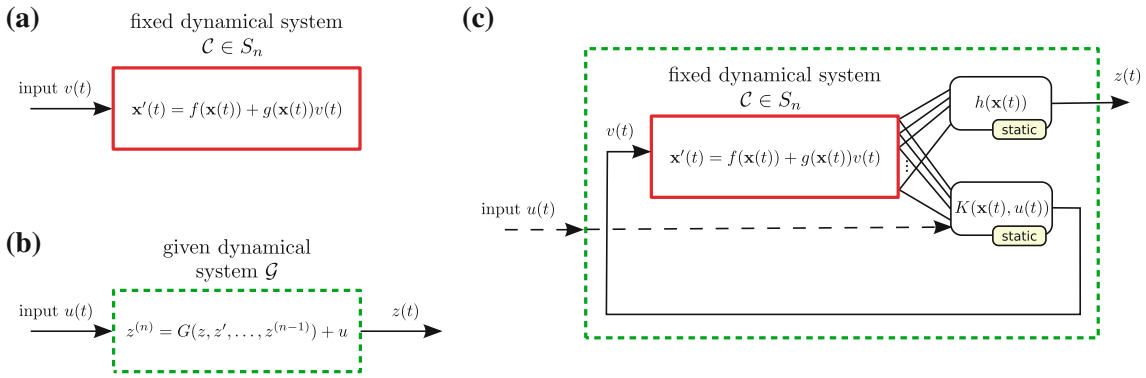

Fig. 1 Computational architectures considered. **a** Fixed dynamical system $\mathcal{C} \in S_n$, which is of the form of Eq. 8. **b** A given arbitrary nth order dynamical system $\mathcal{G}$ (target system) with external input $u(t)$, which should be emulated by system (**a**) using an appropriate static feedback $K(\mathbf{x}(t), u(t))$ and a static readout function $h(\mathbf{x}(t))$. This results in structure (**c**), which emulates the system $\mathcal{G}$, i.e., it delivers the same output as system $\mathcal{G}$, i.e., $h(\mathbf{x}(t)) = z(t)$ for any input $u(t)$

with

$$\mathbf{A}_n = \begin{pmatrix} 0\,1\,0\,\ldots\,0 \\ 0\,0\,1\,\ldots\,0 \\ \vdots\,\vdots\,\vdots\,\ddots\,\vdots \\ 0\,0\,0\,\ldots\,1 \\ 0\,0\,0\,\ldots\,0 \end{pmatrix} \quad \mathbf{b}_n = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}.$$

More generally, one can say that the nonlinear feedback linearizable system $\mathcal{C}$ is *feedback equivalent* to the linear system $\mathcal{L}$. The notion of *feedback equivalence* is an equivalence relation expressing that two systems of differential equations can be transformed into each other through application of a suitable feedback and a change of basis in the state space (Maass et al. 2007; Sontag 1998). Such a change of basis can be achieved by an appropriate readout function. The concept of *feedback equivalence* will serve as a basis for our theoretical model. It will allow us to demonstrate how certain generic physical structures (based on mass–spring systems) can be transformed with an appropriate feedback and a readout in order to emulate a given dynamic target system (i.e., emulate a given computation).

Let us assume that our dynamic target system is $\mathcal{G}$, has the order n, is nonlinear, and is of the form

$$\mathcal{G}: \quad z(t)^{(n)} = G(z(t), z(t)', \ldots, z(t)^{(n-1)}) + u(t)\,, \qquad (3)$$

where $G : \mathbb{R}^n \to \mathbb{R}$ is a sufficiently smooth, but otherwise arbitrary, nonlinear function. We can easily demonstrate that the system $\mathcal{G}$ is also feedback equivalent to the linear system $\mathcal{L}$ of Eq. 2. Indeed, if we chose the state space transformation $x_1(t) = z(t)$, and $x_{i+1}(t) = z^{(i)}$ for $i = 1, \ldots, n-1$ and the feedback $v(t) = G(\mathbf{x}(t)) + u(t)$ we can transform the linear system $\mathcal{L}$ into the nonlinear system $\mathcal{G}$, hence they are feedback equivalent. Since $\mathcal{C}$ is feedback equivalent to $\mathcal{L}$, and $\mathcal{L}$ is feedback equivalent to $\mathcal{G}$, we can conclude that also any system $\mathcal{C}$ is also feedback equivalent to any system $\mathcal{G}$ (for a proof we refer to Maass et al. (2007).[2]) In other words, for a given nonlinear, dynamic system $\mathcal{G}$ (Eq. 3, which represents our computation, which we want to emulate) there exists a feedback $K(\mathbf{x}(t), u(t))$ and a readout $h(\mathbf{x}(t))$ (both nonlinear and static) to transform the nonlinear, dynamic system $\mathcal{C}$ such that it emulates $\mathcal{G}$. To be more specific, for any input $u(t)$ the transformed system (i.e., $\mathcal{C}$ plus feedback) provides through its static, nonlinear readout function $h(\mathbf{x}(t))$ the same output as the original target system $\mathcal{G}$, i.e., $h(\mathbf{x}(t)) = z(t)$ (see Fig. 1).

This implies (see Fig. 1) that one can have a fixed system $\mathcal{C}$ (red box), which operates as a universal module, and one just has to add a suitable feedback and readout in order to emulate any given nonlinear system $\mathcal{G}$ (green dashed-lined box), as long it has the form of Eq. 3. Note that the description of the system $\mathcal{G}$ in Eq. 3 is remarkable general. It includes computations like, for example, to describe any time invariant, memory fading operator (i.e., operators, which can be described by a Volterra Series). This is exactly the class of computation we considered in Hauser et al. (2012). However, as already stated in the introduction, the proposed mathematical model here is more general. It also includes computations to generate nonlinear limit cycles in order to provide trajectories for locomotion. Furthermore, it even allows to describe input-dependent state switching. Actually, the proposed setup is able to emulate any conceivable *continuous* dynamic response to an input stream $u(t)$, hence, one can argue that a system $\mathcal{C}$ becomes a universal computing device for *analog* computing on time-varying inputs by applying an appropriate feedback and readout.[3]

[2] For the sake of completeness we restate the theorem by Maass et al. (2007) in Appendix C by using our notation.

[3] Remarkably, even under noise the proposed computational setup still has maximal possible computational power within the a priori limitation, i.e., that it can emulate any given finite state machine (FSM)—for a proof we refer to Maass et al. (2007). Note that any digital computer is a FSM.

So far this theoretical result has only been applied to neural networks as concrete instantiations for such fixed dynamical systems $\mathcal{C}$, see Maass et al. (2007). Here we show, in the context of morphological computation, both analytically and through computer simulations that parts of the physical body can also be employed as such fixed modules $\mathcal{C}$. The idea is to provide the physical body (or a part of it) with appropriate feedback and to add a readout in order to emulate a given dynamic target system $\mathcal{G}$. The most basic dynamic systems, which are used to describe the dynamics of the compliant body of biological systems or robots, are nonlinear mass–spring systems. Their general mathematical model is

$$\begin{aligned} x_1' &= x_2 \\ x_2' &= -p(x_1) - q(x_2) + \frac{1}{m}v \,, \end{aligned} \tag{4}$$

where $x_1 \in \mathbb{R}$ is the displacement of the spring relative to the resting length l_0, $x_2 \in \mathbb{R}$ the rate of change of x_1 (i.e., velocity x_1'), $m \in \mathbb{R}^+$ the mass and v the sum of all external forces acting on the spring. The functions $p : \mathbb{R} \to \mathbb{R}$ and $q : \mathbb{R} \to \mathbb{R}$ are nonlinear functions, which describe the properties of the spring.[4] In order to have a physically realistic and, therefore, a stable system these functions have to be monotonically increasing and fulfill $p(0) = 0$ and $q(0) = 0$.

Now, in order to show that such a nonlinear mass–spring system can serve as a fixed module $\mathcal{C}$ and, therefore, can be used for morphological computation, we have to demonstrate that it belongs to the class of feedback linearizable systems S_n. This can be done by the applying the theorem described in Appendix B. The actual proof can be found in Appendix D.

This implies that, assuming that a part of the compliant body can be described by the general form of Eq. 4, this body part can be used for a powerful morphological computation device, without altering its physical structure. More specifically, for a given target computation (encoded as nonlinear dynamic system $\mathcal{G}$ in the form of Eq. 3), one simply has to add the corresponding static feedback $K(\mathbf{x}(t), u(t))$ and the corresponding static readout $h(\mathbf{x}(t))$, in order to emulate $\mathcal{G}$. This physical body part is not changed at all. It simply reacts, obeying the laws of physics, on the given feedback $K(\mathbf{x}(t), u(t))$. In order to complete the morphological computation device the static nonlinear readout $h(\mathbf{x}(t))$ is applied. Since, the feedback as well the readouts are static one could argue that the "dynamic" part of the computation has been "outsourced" to the physical body.

Note that in the case of a nonlinear mass–spring system we a have dynamic system $\mathcal{C}$ of order $n = 2$. Consequently, a morphological computation setup using one of such a system can only emulate nonlinear systems $\mathcal{G}$ of order $n = 2$. However, this is not a real limitation, since, as Maass et al. (2007) demonstrated, by combining a number of such basic systems one can still emulate differential equations like $\mathcal{G}$ with an order higher than the one of the spring-mass systems, i.e., $n > 2$. In the case of a real physical body this would be like employing different parts of the compliant morphology for the same computation.

Another example of a valid system $\mathcal{C}$ in the context of morphological computation is a set of *linear* mass–spring systems in parallel. They are all independent subsystems, but they all receive the same input $v(t)$. A corresponding proof[5] can be found in Appendix E. An interesting result of this is that such a combined system has to have diversity in its structure (i.e., the spring properties of the subsystems have to be diverse) in order to be useful at all for the proposed morphological computation devices. This suggests that diversity in the physical properties of a compliant body is in fact beneficiary, and it is interesting to note that a similar kind of diversity is exhibited by biological systems as well.

In the following sections, we will discuss the practical implications of the theoretical results presented here. We will demonstrate how the idea of outsourcing parts of the computation to the body can be taken even further and how we practically implement the proposed morphological computation setup. In addition, we will present results of computer simulation to support or view.

3 Practical implications

So far, we have presented the theoretical basis for morphological computation. Now we are ready to take a look at the practical implications. Assuming that we have a given target system $\mathcal{G}$ (which presents some kind of computation that we want to emulate), naturally, the question arises: What is the corresponding nonlinear feedback $K(\mathbf{x}, u)$ and what is the corresponding nonlinear readout $h(\mathbf{x})$ for a nonlinear mass–spring system to emulate $\mathcal{G}$? We answer this question by showing a specific example. We choose the nonlinear Van der Pol equation as a dynamical target system $\mathcal{G}$, i.e., the system which should be emulated by the morphological computation device. The Van der Pol equations describe a stable, nonlinear limit cycle. Hence, they present an interesting example of a dynamic system, which produces nonlinear, repetitive pattern, which could be, for example, used for locomotion. The differential equations considered are

$$\begin{aligned} x_1' &= x_2 \\ x_2' &= -x_1 + (1 - x_1^2)x_2. \end{aligned} \tag{5}$$

The corresponding function G is found by rewriting the set of two differential equations of order one into one differ-

[4] The scaling factor $\frac{1}{m}$ is already included in the functions $p(x_1)$ and $q(x_2)$.

[5] This proof is analogous to the proof for a set of parallel neurons as presented in Maass et al. (2007)

ential equation of order two. This results in $x_1'' = -x_1 + (1 - x_1^2)x_1'$. By transforming the variables by $z = x_1$ we get $z'' = -z + (1 - z^2)z'$, hence, the corresponding function is $G(z, z') = -z+(1-z^2)z'$. By using the feedback $K(\mathbf{x}, u) = p(x_1) + q(x_2) - x_1 + (1 - x_1^2)x_2$ and, for example, the readouts $h_1(\mathbf{x}) = x_1$ and $h_2(\mathbf{x}) = x_2$ we can use the nonlinear mass–spring system to emulate the van der Pol equations. Note that if only the feedback $K^{\#}(\mathbf{x}, u) = p(x_1) + q(x_2)$ were applied, the system would be linearized and the resulting linear system would be of the form of Eq. 2 with $n = 2$.

As mentioned earlier, an interesting fact is that the feedback K as well as the readout h are static. Hence, they provide suitable targets for supervised learning techniques. These static functions could be implemented, for example, by two feed forward neural networks with one hidden layer, since such networks can approximate any continuous function with arbitrary precision, see Hornik et al. (1989). However, this is only possible if the desired feedback and readout functions are known, and this is only the case if the exact properties of mass–spring system used, i.e., the functions $p(x_1)$ and $q(x_2)$, and the mathematical model of the target system, i.e., $\mathcal{G}$, are known. In general this is not the case and, therefore, no clear target functions are available.

Nevertheless, a different point of view can provide us with a solution. Instead of having nonlinear feedbacks and readouts, one could consider to "outsource" the task to provide nonlinearities to the physical body. This is possible, as we are going to show, if the compliant body is sufficiently complex, nonlinear and diverse. As a consequence only linear (but still static) feedbacks and readouts will be sufficient to emulate complex, nonlinear dynamic systems. Moreover, we will demonstrate, that it is even possible to randomly choose and fix the linear feedback, which then results in a morphological computation setup, in which only some linear output weights have to be adapted. This points to a particularly interesting feature of morphological computation, namely, it facilitates the *learning* of emulating complex computations by reducing the task of learning to emulate complex, nonlinear computations (encoded in nonlinear differential equations) to the much simpler task of finding some static weights. As a consequence learning is much faster and can not get stuck in local minima of the mean-squared error function. In addition, the setup with linear weights has arguably optimal generalization capabilities, see Bartlett and Maass (2003).

In the next section, we explain in more detail why this outsourcing to the body is possible at all.

4 Application of the theory to generic models of complex, compliant physical bodies

A typical morphological structure of a biological organism, or a compliant robot, is complex, nonlinear and highly dynamic. In the context of robot design these properties are typically undesired, since tend to be hard to control. However, it is exactly these properties that will allow us to "outsource" the load of providing nonlinearity directly to the morphological structure and, therefore, employ the physical body for morphological computation.

The underlying idea is to view the morphological structure as some fixed nonlinear *kernel*, which provides us with high-dimensional projections and nonlinear combinations of our input. Hence, the required nonlinearity (next to the dynamics) is provided by the morphological structure itself and, therefore, **linear** feedbacks and readouts are sufficient in order to emulate **nonlinear** differential equations.

The notion of a kernel that we use here is closely related to the notion of a kernel for Support Vector Machines in machine learning. For more details please see Appendix A and Vapnik (1998). Whereas a kernel for a Support Vector Machine is just a virtual mathematical concept, we are considering here concrete physical implementations of a kernel. Ideally a kernel should map any set of different input vectors $x_1, \ldots, x_m \in \mathbb{R}^k$ onto m linearly independent output vectors $z_1, \ldots, z_m \in \mathbb{R}^l$ (because a linear function can assign any given values to linearly independent vectors $z_1, \ldots, z_m$). Whereas a radial basis function (RBF) kernel for a Support Vector Machine satisfies this property for any finite m (which is only possible if $l = \infty$), a physical implementation of a kernel (with some finite number l of output channels) can satisfy this property at best for some fixed finite range of m. But if one chooses l sufficiently larger than m, any randomly connected analog circuit (or some other physical device) consisting of sufficiently many and diverse nonlinear components tends to map a large class of pairwise different inputs $x_1, \ldots, x_m \in \mathbb{R}^k$ onto linear-independent outputs $z_1, \ldots, z_m \in \mathbb{R}^l$. This suggests that the complex morphological structure of biological systems or robots are able to provide such a *kernel*. From this point of view the complexity of the morphology is highly important, since it determines the complexity (and even the type) of computation that can be carried out by the morphological computation device.

In addition, this results in a suprising consequence. While in most classical approaches in robot control this complexity is unwanted and measures such as using rigid body parts and high torque servos are taken to reduce it, our proposed setup requires such high complexity and nonlinearities in the dynamics of the morphological structure.

Note that the theory presented in Sect. 2 is not able to quantify nor to guarantee the kernel property of a given system $\mathcal{C}$. Therefore, we tested in computer simulations to what extent the theoretical predictions hold for physically realistic models of complex physical bodies. More details on these generic models are presented in the next sections.

5 Implementation of mass–spring networks

In order to approximate real, compliant body parts of biological organisms and robots we considered here the implementation of random, recurrently connected networks of nonlinear springs and masses, to which we simply refer as *mass–spring nets*. We will demonstrate with a number of experiments that such generic networks can provide the necessary complexity (i.e., kernel property) in order to emulate interesting nonlinear differential systems related to adaptive periodic movement generation. All presented simulations were implemented in Matlab and were simulated at a time step of 1 ms. In order to keep the simulations simple we implemented the mass–spring networks in a 2D plane, hence, we did not include gravity. However, the presented theory and the resulting approach is not restricted by these constraints.

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.

5.1 Constructing mass–spring networks

The construction of the mass–spring networks was based on following principles: First, the final network should be realizable as a real physical system. Second, it should approximate the dynamics of compliant parts of real robots and organisms, and third, it should be generic, i.e., not be constructed for any specific task.

A fixed number of N nodes (mass points) were randomly positioned (uniformly distributed) within a defined range of a 2D 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. 2. Every single nonlinear spring of such a network can be described by Eq. 4. Note that the nonlinearities of the springs are desired to enhance the kernel property of the network. However, also the geometric assembly of such networks contributes nonlinearities due to the nonlinear relationship between the spring forces and the actual forces acting on the masses (i.e., due to the influence of the angles).

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 $\mathbf{x} = [0, 0]^T$ and, therefore, all masses were at rest). In order to accomplish this we set 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,[6], had to be monotonically increasing and fulfill $p(0) = 0$ and $q(0) = 0$. Typically, they are modeled by third-order polynomials, see, e.g., Palm (1999). 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. 4, the springs should be diverse. 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.

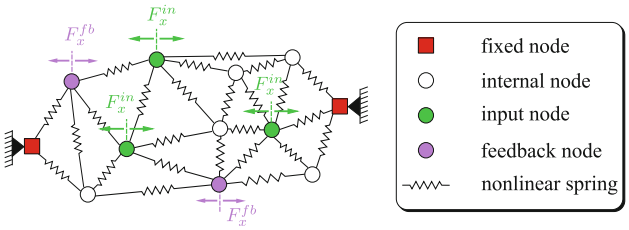

Fig. 2 Schematic example of a generic mass–spring network. The nodes (masses) are connected by nonlinear springs. The *red* nodes are fixed in order to hold the network in place. The *green* nodes are randomly chosen inputs nodes, which receive the input $u(t)$ in form of horizontal forces scaled by randomly initiated weights. The *purple* nodes are feedback nodes. Similarly, they receive the output $y(t)$ in form of a scaled, horizontal force. Compare to Fig. 3a

The left most and the right most mass nodes were fixed in order to keep the network in place (squared, red nodes in Fig. 2). A certain percentage of points were randomly chosen to be input nodes (green nodes in Fig. 2). During simulation they received a linearly scaled version of the current input in form of a horizontal force. Note that the application of the input in form of a horizontal force is arbitrary. We tested a number of different input forms and they all worked. In the biological context the input force could be coming from, e.g., the impact during ground contact.

Before the simulation started the input scaling factors (weights $\mathbf{w}_{\text{in}} = [w_{\text{in},1}, w_{\text{in},2}, \ldots]^T$) had been randomly drawn from a certain range and had been fixed subsequently. In the same way, before the simulation started, the feedback nodes had been randomly chosen. They received during simulation a linearly scaled version of the current output in form of horizontal forces. The corresponding feedback weights were denoted by $\mathbf{w}_{\text{fb}} = [w_{\text{fb},1}, w_{\text{fb},2}, \ldots]^T$. Similar to the input weights they were randomly initialized and subsequently fixed.

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 ($\mathbf{w}_{\text{out}} = [w_{\text{out},1}, w_{\text{out},2}, \ldots$

[6] 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 LaSalle's Theorem [see Theorem 4.4 and Corollary 4.2 in Khalil (2002)].

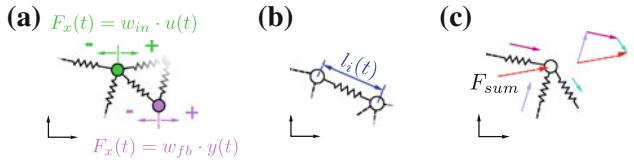

Fig. 3 Details on the implementation of the simulation of the mass–spring networks. **a** The input is applied to an input (*green*) node as a horizontal force F_x proportional to the input signal u (scaled by a randomly initialized weight w_{in} for this input node). The feedback was implemented similarly with a force proportional to the output y scaled by w_{fb} (*purple*). **b** The output of the systems 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, the feedback as well as the output can be multi-dimensional. **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

$\ldots, w_{\text{out},L}]^T$), in contrast the rest of the network parameters, were adapted in the learning process.

In general the networks had multiple inputs, outputs and feedback loops. Accordingly, the corresponding matrices were denoted by $\mathbf{W}_{\text{in}}$, $\mathbf{W}_{\text{fb}}$ and $\mathbf{W}_{\text{out}}$.

5.2 Simulating mass–spring networks

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

$$m p_x'' = F_x + w_{\text{in}} u + w_{\text{fb}} y \tag{6}$$

$$m p_y'' = F_y \ , \tag{7}$$

where p_x'' and 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 spatial dimensions, $w_{\text{in}} u$ was the weighted input, and $w_{\text{fb}} y$ the linear feedback. Note that the input as well the feedback were defined as horizontal forces (see Fig. 3a) and if the mass point was no input node $w_{\text{in}} := 0$, accordingly for the feedback $w_{\text{fb}} := 0$. For the sake of simplicity [7] 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 $x_2 = (x_1(t) - x_1(t - \Delta t))/\Delta t$ 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 (see Fig. 3c). 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. 6 and 7. If the mass point was an input node the current input $u(t)$ was added in form of a scaled horizontal force (see Eqs. 6, 3a). Accordingly, if the mass point received feedback the corresponding force, i.e., $w_{\text{fb}} y(t)$, was added too. The new position and velocity of the mass were found by numerically integrating Eqs. 6 and 7 (fourth-order Runge–Kutta). 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. 3b).

5.3 Learning the linear readout of a mass–spring network

The structure of the mass–spring network, the input and feedback weights, 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 adjusted. The learning process was carried out with open loop, i.e., instead of the real output the target signal y^* was fed back (Fig. 4a for the general case of multiple inputs and outputs). Thus, the system was forced into the desired operative state by a "teacher" signal. Hence, this setup is referred to as *teacher forcing*. After the learning process the loops were closed and the system ran freely (Fig. 4b). Note that in this case already small perturbations (for example numerical imprecisions in the simulations or noise in real systems) would lead such a closed loop system away from its learned trajectories. Therefore, we superimposed during the learning process the outputs with white noise ν. Thus, the found output weights did not simply reproduce the desired target trajectories, but rather were able to do so even under the influence of noise. Note that alternatively one could use methods of regularization, see Bishop (1994). As a result of the noise, the learned trajectories were robust. Note that this is a remarkable fact. Noise is inherent to any real-world application and a lot of time unwanted, where in our case it is a desired property, since it contributes to the robustness.

For learning we considered a network of N nodes connected by L springs. During the teacher forcing phase we collected the current lengths of every single spring $l_i(t)$ for $i = 1, \ldots, L$ 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

[7] 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 loss of generality. Nevertheless, in a real biological body (or robot) a diversity of masses is natural and it contributes further diversity.

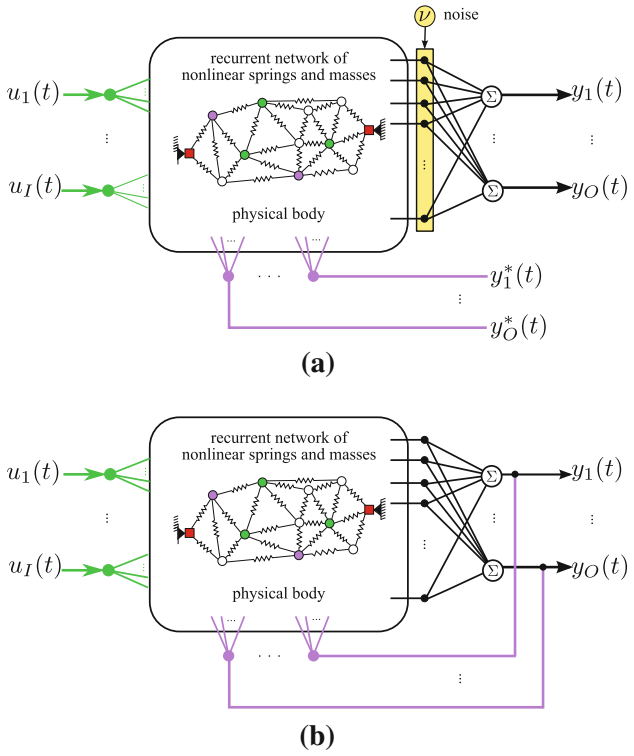

Fig. 4 The learning and exploitation setups. **a** The learning procedure (teacher forcing) with superimposed white noise ν. **b** Final setup after learning, when the loops are closed and the system runs freely

$\mathbf{w}^*_{\text{out}} = \mathbf{L}^\dagger\mathbf{T}$, with $\mathbf{L}^\dagger$ being the (Moore–Penrose) pseudoinverse, since in general $\mathbf{L}$ was not a square matrix. Note that the same procedure could be applied in the case of multiple inputs and/or multiple outputs (feedback loops), with a matrix $\mathbf{W}^*_{\text{out}}$ of optimal output weights. Multiple inputs are interesting for emulating computations, which, for example, incorporate various input streams from different sensors. Multiple outputs are useful, e.g., in the context of locomotion to produce motor commands for different degrees of freedom.

6 Results of computer simulations

In this section, we provide a number of results of computer simulations in order to demonstrate the applicability of our approach. We constructed generic mass–spring nets, as described in the previous section, and employed them for computational tasks relevant for robots.

6.1 Generating stable, nonlinear limit cycles with morphological computation

The general description of dynamical systems (Eq. 3), which can be emulated by the proposed morphological computation setup, also include nonlinear limit cycles (e.g., the Van der Pol equations as we have shown in Sect. 3). Nonlinear limit cycles are very appealing for the control of robots, since they represent an elegant way to describe repetitive patterns, which are typically used for locomotion. A standard approach to implement such limit cycles is to use a nonlinear oscillator (i.e., central pattern generators, CPGs) or a network of such oscillators, e.g., as in Righetti and Ijspeert (2008). As already discussed in the introduction, such CPG networks are generic and exhibit the property of robustness. However, the parameters of such networks are typically hand-tuned or are found by computationally intensive nonlinear search algorithms, e.g., genetic algorithms. Our theory suggests that nonlinear mass–spring systems and, therefore, the body of a robot itself, can be used to generate autonomously and robustly such limit cycles. Furthermore, by employing the nonlinear dynamics of the morphology, the problem of finding valid parameters can be reduced to a simple linear regression.

In order to demonstrate that our proposed morphological computation setup is able to generate robust limit cycles we chose three different limit cycles as tasks. They are summarized in Fig. 5. The first column: (a, e) and (i) shows the differential equations describing the limit cycles. The second column: (b, f) and (j) shows the corresponding trajectories of the state variables x_1 and x_2 over time, and the third column: (c, g) and (k); in the state space (i.e., x_1 vs. x_2). These two state variables are the outputs, which should be produced autonomously by the morphological computation device, i.e., the mass–springs systems and two linear readouts.

The first limit cycle (first row) has been previously introduced in Sect. 3. It is the dynamic system of the Van der Pol equations (Fig. 5a). The second one (second row) is an example taken from Khalil (2002). Due to its quadratic terms $(x_1^2+x_2^2)$ we refer to it as the "quadratic" limit cycle (Fig. 5e). The third example (Fig. 5i) is an artificial trajectory, defined by two sinusoidal functions with a frequency ratio of $f_1/f_2 = 3/2$. Together (x_1 vs. x_2) they produce a complex Lissajous figure, which is a trajectory with multiple crossings (see Fig. 5k). Note that any dynamical system, which should emulate the Lissajous trajectory, must have an order higher than two. As already argued in the theory Sect. 2, although the basic system (the single nonlinear mass–spring system) is only 2D, by connecting various basic systems, it is possible to emulate systems of higher order too (which we will demonstrate by this example).

As already discussed in the theory Sect. 2 the employed morphological structure (i.e., the body) is fixed and only the readout and the feedback determine, which nonlinear system is emulated. This implies that we can use *one* generic mass–spring network (denoted here as the *general* network) for *all* three limit cycle tasks. It was found by the previously described random process and it is depicted in Fig. 5m. The number of mass points was $N = 25$ and the number of connecting springs was $L = 61$. The squared, red nodes

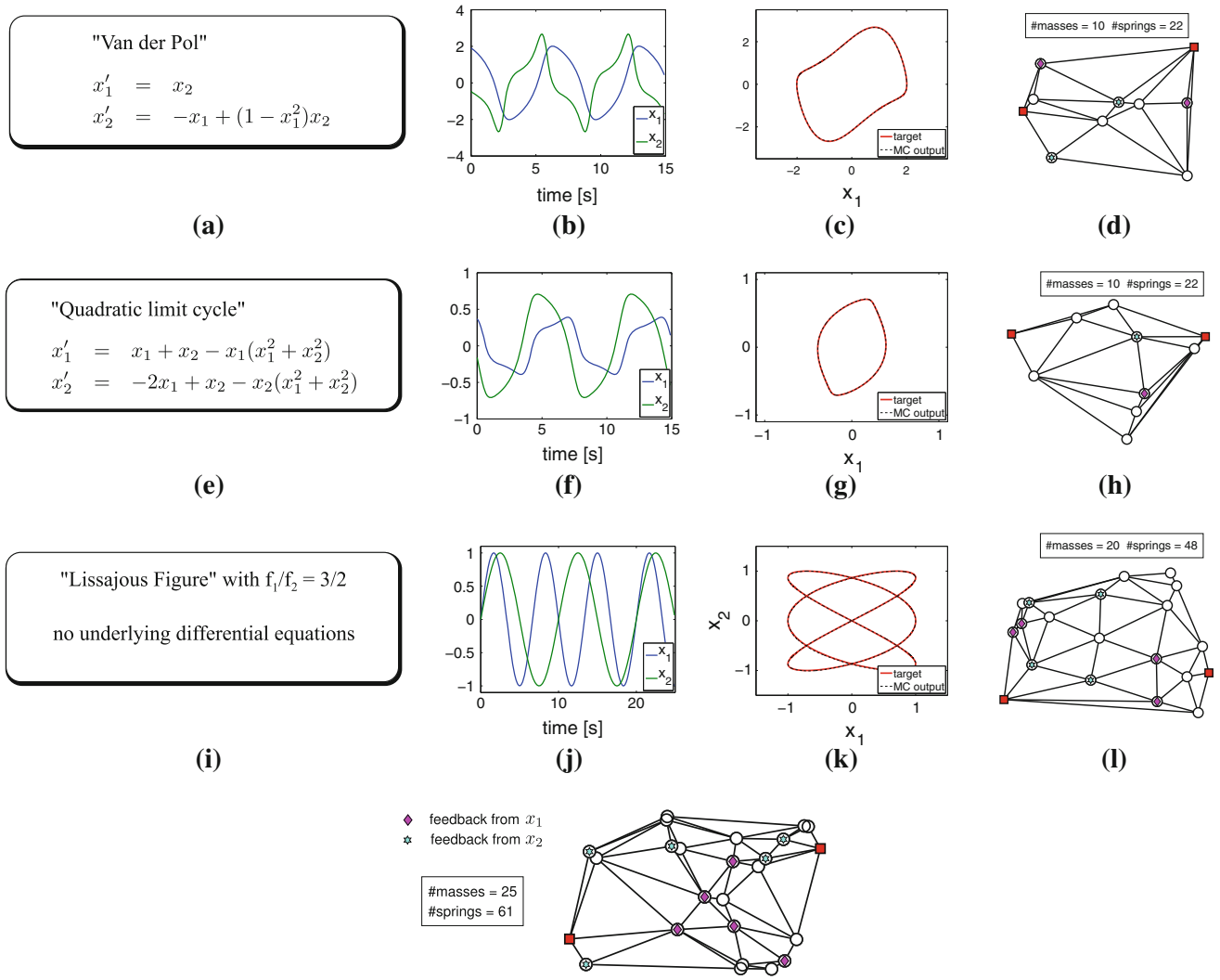

Fig. 5 Results for three different tasks, i.e., three different limit cycles, which were chosen to be autonomously generated through morphological computation by a generic mass–spring network with linear feedbacks. The first *column* shows the differential equations describing the three limit cycles. The second *column* shows the corresponding trajectories of the state variables, i.e., x_1 and x_2 (target signals). The third column shows the same state variables in the phase plane (i.e., x_1 vs. x_2). The *red line* is the target signal and the *black dottend line* the output of the morphological computation devices. The fourth *column* shows the specialized networks, used to produce the limit cycles. They are remarkably simple. It is also possible to produce the three limit cycles with only one (general) network, e.g., the one of (m)

are fixed mass points. The connections are nonlinear springs with randomly drawn values for the spring and damping functions (i.e., k_1, k_3, d_1 and d_3 of the functions p and q). The randomly chosen nodes, which received feedback from the first output (x_1), are marked by purple diamonds. The randomly chosen nodes, which received feedback from the second output (x_2), are marked by aqua marine hexagrams. The corresponding linear feedback weights $\mathbf{W}_{\text{fb}}$ were randomly initialized and subsequently fixed. Note that there were no input nodes, since, after learning the morphological computational device should generate autonomously the desired trajectories without any external stimulation. The output was defined as previously described by a weighted sum of all 61 current spring lengths. The optimal outputs weights $\mathbf{W}^*_{\text{out}}$ were found by the previously described learning process in Fig. 4a (teacher forcing with superimposing noise with an amplitude of 1×10^{-3} at the spring lengths). For training we used between 15.000 and 30.000 data points (i.e., 15–30 simulated seconds), which correspond to 2–4 times the period of the signal depending on the target. Note that for very short training periods the results was either an unstable system or a limit cycle different from the target. For more details on the chosen ranges for the various parameter we refer to the supplementary material.

After the learning phase the networks were simulated with closed loops, i.e., the outputs were fed back (as in Fig. 4b). The third column of Fig. 5c, g, k shows the corresponding outputs in phase plane, i.e., x_1 vs. x_2. The red lines are the target trajectories of the limit cycles and the black dotted lines are the trajectories produced by the morphological computation device (MC). Note that the readouts as well as the feedbacks were static and linear. Hence, the necessary dynamics as well as the nonlinearities were "computed" by the morphological structure. Since the presented mass–spring networks are simulations of real physical systems, one can conclude that the corresponding real physical bodies can perform the same morphological computations.

The used network of Fig. 5m performed well for all three tasks. However, since the construction process was based on a probability distribution (i.e., based on values, which were randomly drawn from defined ranges) some randomly constructed networks performed better (or worse) for the different tasks than others. Therefore, is was possible to select networks, which performed individually better for one of the three tasks, but performed worse on the other two tasks. Since the complexity of the task was reduced (i.e., one limit cycle instead of three different ones), the networks were typically smaller. The chosen networks are depicted in the last column of Fig. 5. The network for the Van der Pol task had $N = 10$ masses and 22 springs (Fig. 5d). The network for the quadratic limit cycle and $N = 20$ and $L = 22$ (Fig. 5h) and for the Lissajous figure task $N = 20$ and $L = 48$ (Fig. 5l). Note that, for example, the network, which is employed to emulate the Van der Pol equations, is suprisingly simple. This is especially surprising if one takes into account that the morphological computation device not only produced the limit cycle, but rather generated, due to the use of the noise during learning, a robust limit cycle with a region of attraction (of unknown size). This can be concluded from the fact that although numerical imprecision (as inherent to any simulation) was present at any time the systems stayed robustly on the nominal trajectories. We tested all presented networks regarding their stability on the long run. We simulated each network for 1 million time steps (i.e., 1, 000 simulated seconds). The networks stayed on the desired trajectories.

Even more remarkably is the fact that the morphological computation devices were not only robust against small perturbation (e.g., introduced by numerical imprecision) but were rather highly robust. In order to demonstrate this import property we conducted a number of experiments, which are described in the following sections.

6.1.1 Testing the stability of the learned limit cycles

The amplitudes of the noise during the teacher forcing (learning) process were much bigger than the amplitude of the noise introduced by numerical imprecision. Therefore, it is sensible to assume that the morphological computation device is not only able to counteract underlying noise with a small amplitude, but rather is robust towards all kind of disturbances. In addition, for simple mass–spring networks like of Fig. 5h or 5d, the inherent stability of such physical systems (real physical mass–spring systems loose energy over time, e.g., due to friction) can contribute further to stability. In order to demonstrate this stability we conducted various experiments, where we disturbed a mass–spring network with different types of perturbations. For all stability experiments we used the network of Fig. 5h (i.e., the specialized network for the quadratic limit cycle). However, similar results can be obtained for other networks and tasks too. Figure 6 summarizes the results of the conducted experiments.

In a first test (Fig. 6a, b) the system started unperturbed. Suddenly, at $t = 10$ s (start of the red region), instead of the actual produced output x_1 (blue dotted line), the last correct value $x_1(t = 10) = 0.09$ (solid blue line) was fed back for the next 10 s (red region). For a real robot this situation correspond to the case when, for example, one degree of freedom were stuck or if there were a temporal sensor failure for this particular variable.

After some time (at $t = 20$ s; end of the red region) the actual output value of x_1 was fed back again. Figure 6a shows the trajectories of both outputs x_1 and x_2. The red region depicts the time window, when x_1 was locked. Figure 6b shows the same trajectories but in the phase plane. Note that the different colors encode the corresponding time windows as labeled in Fig. 6a. The morphological computation device was able to find back to the desired trajectory after the disturbance had vanished.

In a second stability test (Fig. 6c, d) all nodes received from 10 to 20 s (red region) a constant horizontal force. The amplitudes were uniformly drawn from the range $[-10, +10]$. After $t = 20$ s the disturbing input vanished suddenly and the system ran freely again. Figure 6c, d shows the trajectories of the outputs x_1 and x_2. Again, colors used in the phase plane correspond to the colors of the labels of the different time windows in Fig. 6c. Remarkably, although the perturbation was fairly strong, the system was able to recover from it and to find its way back to its nominal trajectory. Note that the perturbation led the trajectories to a region in state space "far away" from the area, which had been covered by the noisy learning data. Hence, one would conclude that the system was able to generalize to values outside the range of the presented learning data. However, this is not entirely true. Assuming that we start both systems, the original dynamic system and the device, which emulates this system, at the same point in the state space $\mathbf{x} = [x_1, x_2]^T$, in general, the trajectories of the two systems, which lead them back to the nominal limit cycle, will differ. Nevertheless, both will come back to the desired trajectory.

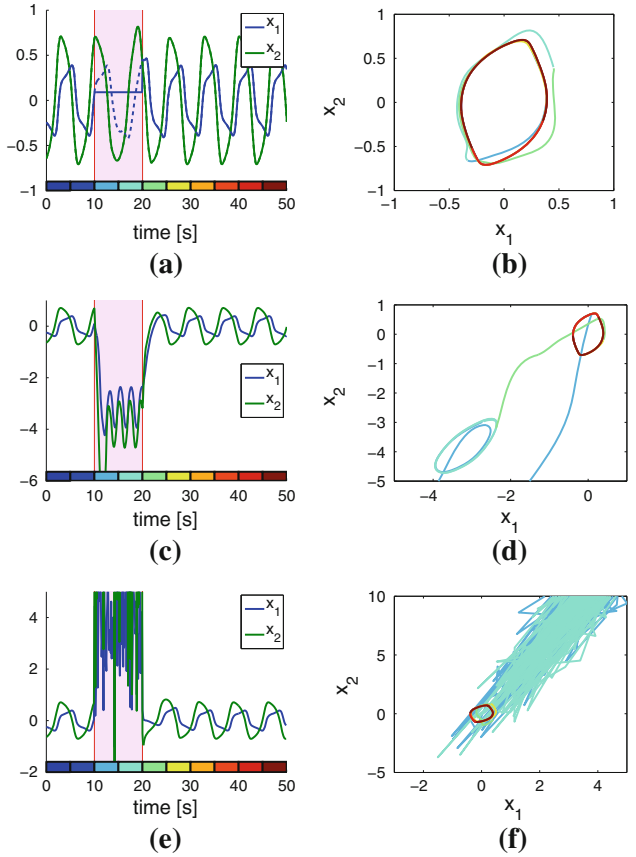

Fig. 6 Experiments to demonstrate the stability of the learned limit cycle using morphological computation. The used network was the one depicted in 5h ("quadratic" limit cycle). The red regions mark the time windows, when the perturbations appeared. The color coding of the plots of the second column correspond to the labels of the time windows in the plots of the first column. Following three perturbations were tested: **a**, **b** The output x_1 was held at a constant level (*blue solid line*) and fed back into the system instead of the actual output (*blue dotted line*). **c**, **d** All nodes received a constant force, i.e., the whole network was distorted. **e**, **f** The measured current lengths $l_i(t)$ were superimposed by white noise (for a real robot this corresponds to a situation with noisy sensory signals). Hence, the outputs and the feedbacks were noisy too. In all three cases the system was able to recover and to find its way back to the nominal limit cycle trajectory

Therefore, for practical reasons, one could say that the morphological computation device is able to emulate the same limit cycle with a stability similar to the one of the original system.

A third experiment was conducted in order to show that the setup was also able to recover from stochastic disturbances (Fig. 6e, f). In the time window from 10 to 20 s the signals of the sensors, which read the current lengths of the nonlinear springs, i.e., $l_i(t)$ for $i = 1, \ldots, L$, were superimposed by white noise. Consequently, there were noisy outputs and, therefore, noisy feedbacks. After 20 s, when the noise had vanished, the system recovered fast to the correct trajectories.

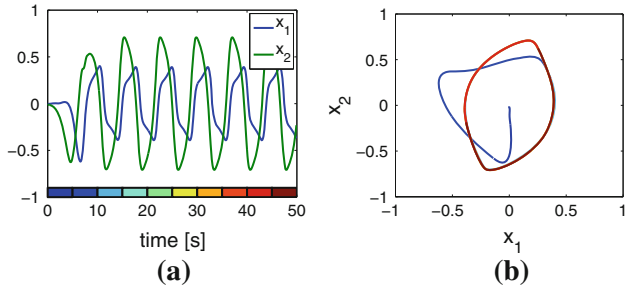

Fig. 7 Recovery of the morphological device from the state of total rest, i.e., *swing-up*. **a** shows the trajectories over time and **b** shows the same trajectories in phase plane. The *color* coding of (**b**) corresponds to the color labels of the time windows in (**a**)

6.1.2 Recovering from the state of total rest: swing up task

In an additional experiment we demonstrate that the system was even able to recover from the state of total rest. For this reason we started the network (same network as previously used—Fig. 5h) in a state, where all velocities and accelerations of all masses were zero and the nonlinear springs had the exact lengths of their resting lengths. Hence, if no external force would have been applied (e.g., open loop) the network would have stayed at rest. However, we closed the loop and the system swung up to the desired trajectories. Since the outputs were based on a weighted sum of the actual lengths, which are non zero (even at $t = 0$), the outputs, and, therefore, the feedbacks, were nonzero and, hence, the system was able to recover from its initial state. The results are summarized in Fig. 7a, b.

6.2 Generation of different walking patterns using the same mass–spring network

As we have seen in the first experiments one physical body (i.e., mass–spring network) can be used to produce different nonlinear patterns. This is especially interesting in the context of locomotion. A lot of animals display different gaits in order to be more energy efficient and to adapt to changes in the environment. Note that the body of the animal does not change, but it is still able to produce such diverse nonlinear patterns (i.e., gaits). This is exactly what our approach offers. The physical body serves as some basic dynamic module, and only the readout and the feedback (both static) define what system is emulated. Hence, different static feedbacks loops (i.e., different output weights $\mathbf{W}_{\text{out}}$ and feedback weights $\mathbf{W}_{\text{fb}}$) can force the same physical body to produce different patterns. Furthermore, since in our setup the feedbacks were randomly initialized and subsequently fixed, we will demonstrate that already different readout weights $\mathbf{W}_{\text{out}}$ are sufficient to produce different dynamic patterns.

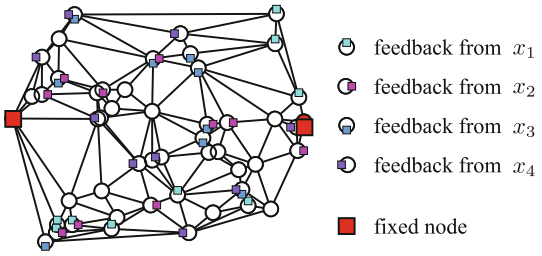

Fig. 8 Generic network used to produce four different gaits for a generic quadruped with the same physical body. The network had four outputs (and feedbacks) corresponding to the four legs. The number of mass points was $N = 50$ and the number of nonlinear springs $L = 137$

The task was to produce four different walking patterns, namely walk, trot, pace and bound, for a generic quadruped using the same morphological structure.

Righetti and Ijspeert (2008) used a network of four coupled oscillators with four different couplings in order to produce four different gaits.

We will demonstrate that it is possible to generate the same rhythmic patterns with one fixed physical body (i.e., mass–spring networks) with some fixed feedback weights, but four different, linear readouts.

The used mass–spring network can be seen in Fig. 8. It consisted of 50 masses and 137 nonlinear springs. It was constructed as previously described in Sect. 5. For more details on the construction parameters used we refer to the supplementary material. Note that the network had four outputs (denoted here by x_1, x_2, x_3 and x_4) and, therefore, it had four corresponding feedback loops. The different, colored squares mark the randomly chosen nodes, which received feedback. Some of the nodes received feedback from various outputs. The linear feedback weights were randomly initialized and were subsequently fixed. Only the linear readout weights were adapted in the learning process.

The learning data (targets) was produced by simulating the original equations used in Righetti and Ijspeert (2008) at a time step of 1 ms. The learning procedure was the same as previously described. Every walking pattern (walk, trot, pace, and bound) was simulated independently from each other. Finally, we had four output matrices of the size 137×4, one for each gait, i.e., $\mathbf{W}_{\text{walk}}$, $\mathbf{W}_{\text{trot}}$, $\mathbf{W}_{\text{pace}}$, and $\mathbf{W}_{\text{bound}}$.

After the learning process we tested the found output weights. Figure 9 summarizes these results. Figure 9a shows the four trajectories, which were produced, when the walking gait matrix $\mathbf{W}_{\text{walk}}$ was used to close the loop. The red dotted lines are the target patterns, and the solid lines are the produced outputs. The next plot (Fig. 9b) shows the output of the morphological computation device, when the loops were suddenly closed at $t = 0$ with the matrix to produce the trot pattern ($\mathbf{W}_{\text{trot}}$) instead of $\mathbf{W}_{\text{walk}}$, i.e., the readout switched from walk to trot.

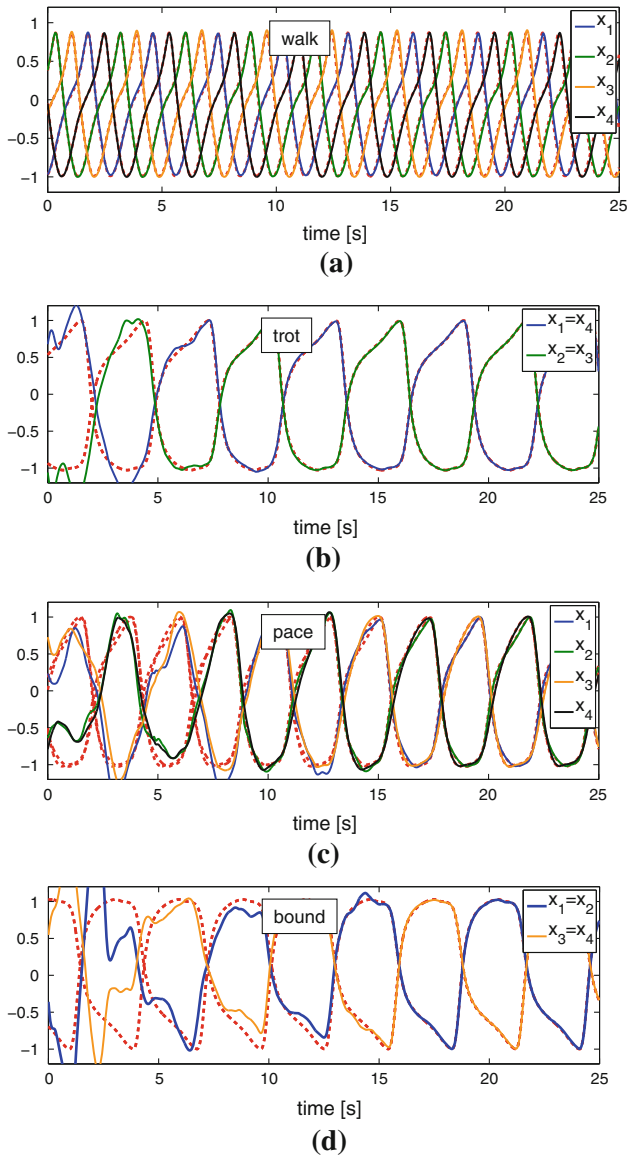

Fig. 9 Generation of different walking patterns for a generic quadruped robot by the use of one morphological structure. **a** walking pattern produce by the morphological computation device.**b–d** Responses of the device, when at $t = 0$, the weights of the outputs suddenly changed from $\mathbf{W}_{\text{walk}}$ to $\mathbf{W}_{\text{trot}}$, $\mathbf{W}_{\text{pace}}$ and $\mathbf{W}_{\text{bound}}$, respectively. The red dotted lines are the target patterns. In all cases, after some transition time, the setup was able to produce the desired patterns

After some transition time the system settled to the desired trot pattern. Figure 9c, d shows the corresponding responses, when it switched suddenly from walking to pace and walking to bound. Again, after some transition time, the morphological computation device produced the desired walking pattern. Note that the different walking pattern had different frequencies, forms and amplitudes. Nevertheless, only different static, linear readouts (or sudden switches between them) were sufficient to force the mass–spring to produce robustly the desired nonlinear patterns.

6.3 Generation of different limit cycles depending on an input signal

In the previously presented examples mass–spring networks were employed to generate autonomously different rhythmic output patterns. Our theory suggests that the proposed setup can emulate even more complex dynamical systems, e.g. systems, which are input driven. This input could be some sensory input stream or an internal control signal. In the context of locomotion this could be, for example, a system, which generates different gaits depending on the desired velocity (i.e., the input).

For the following task we adapted the previously used equations of the "quadratic" limit cycle of Fig. 5e. We added a parameter ε (i.e., the input to the system). Changing this input allows us to change smoothly the shape of the limit cycle. Figure 10 summarizes the properties of the used limit cycle. Figure 10a shows the dynamic target system with the input ε (colored in red). The influence of the input ε on the trajectories of x_1 and x_2 can be seen in Fig. 10b. It shows the limit cycles for three different input values $\varepsilon = 5$, $\varepsilon = 1$ and $\varepsilon = 0.2$ in phase plane. The corresponding trajectories in time can be seen in Fig. 10c–e. Note that the amplitude, the shape as well as the frequency change in dependence of the input ε. Figure 10f, g depicts this fact by plotting the amplitude and the frequency versus the input ε for the state variable x_1.

For the simulation a mass–spring network of $N = 100$ nodes and $L = 283$ connecting springs was used. The network was constructed as previously described in Sect. 5. The target signals were produced by varying the input ε from 0.1 to 5 over the time of 200 s. For more details on the learning data as well on construction parameters used we refer to the supplementary material.

After learning, the morphological computation device was tested. Figure 11 summarizes the results. At the end of the learning process the input was $\varepsilon = 5$ and, therefore, the mass–spring network was in the state to reproduce the corresponding limit cycle of Fig. 10c. The two plots of Fig. 11a show the response of the system, when the input was kept at this constant value of 5. The left plot shows the trajectories of the outputs x_1 and x_2 over time, and the right plot shows the corresponding trajectory in the phase plane.

In the second row (Fig. 11b) the response of the system can be seen, when the input suddenly (at $t = 0$ s) changed from $\varepsilon = 5$ to a constant value of 1. Hence, the system should generate the corresponding limit cycle of Fig. 10d. After some transition phase, the system settled down to the desired limit cycle. After that it stayed robustly on the desired trajectories.

The last row (Fig. 11c) shows the response of the device when the input suddenly changed from 5 to 0.2. Again, after some transition time, the morphological computation device delivered robustly the desired limit cycle.

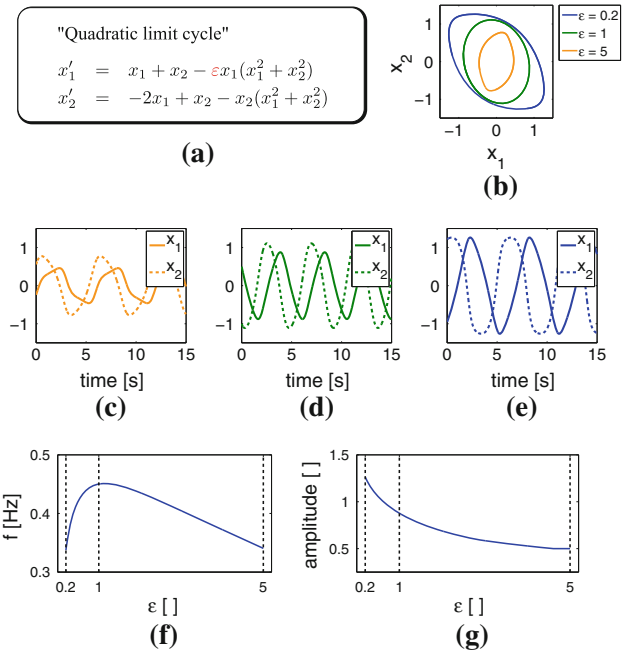

Fig. 10 The target system and the corresponding target limit cycles used for the input dependent limit cycle generator task. **a** Equation of the dynamical target system with the input ε. **b** Different limit cycles for the inputs $\varepsilon = 5$, $\varepsilon = 1$, and $\varepsilon = 0.2$ in phase plane. **c–e** The corresponding trajectories in time. **f** and **g** change of amplitude and frequency in dependence of the input ε

Let us make some remarks here: First, the found readout weights (and the whole feedback loop) were **linear** and **static**. Hence, the nonlinear dynamics, which were apparently involved in this task, were all provided by the generic physical body. Second, in contrast to the previous task of producing walking patterns (Sect. 6.2), here the same static feedback loop (i.e., readout plus feedback) was used to produce different types of limit cycles. This also implies that only the input "decided" what output should be produced. Third, the input was applied as some randomly weighted, but constant forces acting on some randomly chosen input nodes.

Therefore, such a constant input force can be seen as *squeezing* the network a certain points (the ones, which have been randomly chosen to be input nodes). Figuratively speaking, the network produced different limit cycles, depending how strong it was compressed.

7 Discussion

We have introduced a mathematical framework for the analysis of morphological computation with feedback in compliant bodies. In contrast to the case without feedback, as in Hauser et al. (2012), morphological computation with feedback is not limited to the emulation of filters with fading

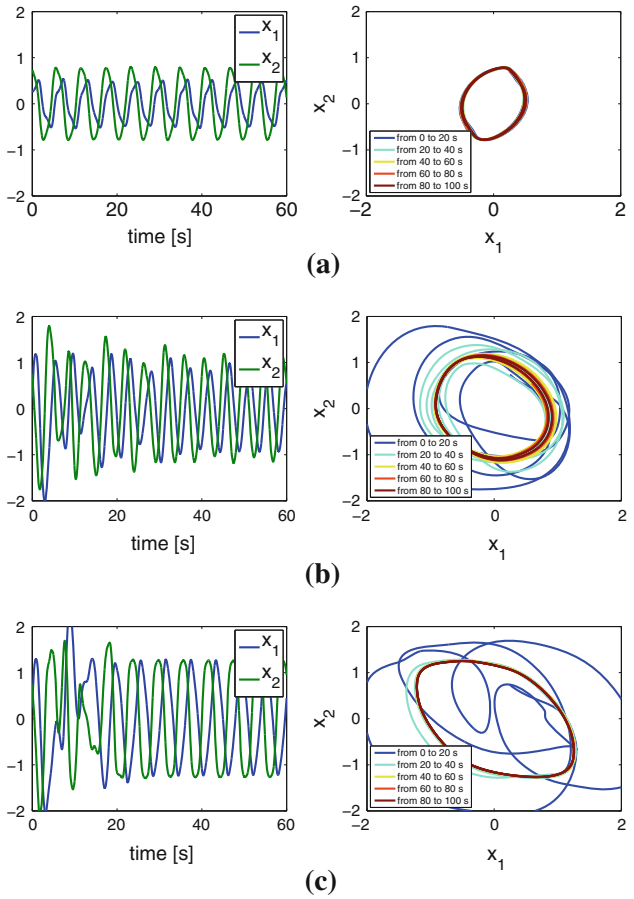

Fig. 11 Testing the morphological computation device to emulate different limit cycle depending on the input. The *left column* shows the two output variables variables x_1 and x_2 evolving over time. The *right column* shows the corresponding trajectories in the phase plane for three different cases. In any of the cases the network started in the state to produce the limit cycle with $\varepsilon = 5$. Case **a**: The input was kept at 5. **b**: The network suddenly received at $t = 0$ instead $\varepsilon = 5$ a constant input of $\varepsilon = 1$. The system changed its output to the corresponding limit cycle. Case **c**: The input changed suddenly at $t = 0$ from 5 to 0.2 and produced, after some transition time, the corresponding limit cycle

memory. In fact, the presented theory shows that there is no longer any a priori limitation on the computational power of morphological computation, if arbitrary (static) continuous function can be used for feedback and readout. We have also discussed that this, however, introduces practical problems, since in a real robot the exact dynamical models of the bodies (especially if their are complex) nor the needed computation for a given task are known. However, heuristic theoretical arguments [as used in Hauser et al. (2012)] suggest that continuous readouts and feedbacks can be approximated by linear functions if the morphological computing device (e.g., a compliant robot) implements a sufficiently complex, nonlinear projection of incoming signals into a high dimensional space. Naturally, a restriction to linear feedbacks and readouts is desirable, because their parameters can be learnt more easily. However, at present the available theory does not offer a prediction about system performance if theoretically ideal feedbacks and readouts are only approximately realized. Therefore, we have tested in computer simulations the performance of concrete morphological computing systems (i.e., random networks of springs and masses) with linear feedbacks and readouts, whose parameters were determined through a simple supervised learning process. We found, that also with linear feedbacks and readouts these morphological computing systems were able to carry out a large range of complex computations, and were not limited by a fading memory. The same type of spring mass systems was able to generate a large repertoire of periodic signals, and it was also able to modify this periodic signal in dependence of some scalar control signal. Furthermore, this pattern generation turned out to become (through training) remarkably noise robust, and resilient even to major disturbances of the system. Hence, morphological computation with linear feedbacks and readouts provides an interesting new paradigm for the generation and switching of a repertoire of periodic signals, as they are needed for locomotion of robots and biological organisms. Recent results, on somewhat similar experiments with artificial neural networks (instead of systems of springs and masses), suggest that morphological computing systems may even be able to learn all this without the help of a supervisor (that provides the desired target output signal), but through autonomous exploration guided only by information when the resulting system performance became better, see Legenstein et al. (2010), Hoerzer et al. (2011, 2012). Further research will have to investigate to what extent such much simpler learning process, which requires no a priori knowledge of the target signal that is to be generated, can also be used to guide morphological computation with feedback.

We demonstrated with various simulations of abstract, randomly connected networks of nonlinear springs and masses that our proposed setup is applicable for a number of tasks relevant for robotics.[8] We propose that compliant bodies can be approximated by such generic networks. Hence, it is tempting to directly implement our proposed scheme for morphological computation with feedback in real (compliant) robots. As our presented experiments imply, a particular interesting application would be to exploit the nonlinear characteristics of the physical bodies of compliant robots in order to produce different types of attractor states (limit cycles), e.g., for different gaits. Especially, the results from the experiment described in Sect. 6.3 (input-dependent generation of limit cycles) represent an interesting case. External inputs (forces) could in principle be used to switch between different attractor states (e.g., gaits). For example, if the robot has to carry a heavy load, naturally, the compliant body is deformed

[8] Any computation that can be encoded in the form of $\mathcal{G} : z(t)^{(n)} = G(z(t), z(t)', \ldots, z(t)^{(n-1)}) + u(t)$ (Eq. 3) is a potential target system.

by the forces introduced by gravity, which could then lead the whole morphological computation device to switch to a new attractor state. Note that this would not require a change in the parameters of (linear) feedbacks or readouts. Actually, there is no *higher control* involved at all. The physical body itself, obeying the laws of physics, simply reacts to the input and the "program" encoded in the static, linear readouts and feedback loops.

This means, we don't have to control all the details (e.g., the joint angle trajectories) of a walking or running robot, but rather only some global parameters in order to "guide" the dynamics. This represents an extremely efficient way of behavior control by exploiting mechanisms of self-organization and exploitation of passive dynamics.

Note that the proposed approach is very generic and there are many possible ways to apply the input to the body. For example, it could be encoded as forces, which occur due to ground contact, or collision with obstacles (e.g., bending a leg), or it could stem from internal model-based signals (i.e., desired velocity), etc.

Another remarkable conclusion can be drawn for future robot design. Typically, nonlinearity and compliance, which are inherent properties of biological systems, are seen, at least from an engineering perspective, as undesirable features for robots. This is mostly based on the fact that such complex systems are very difficult to control. As a result, classical robot designs are based on rigid body parts and high torque servos in order to suppress unwanted dynamics and to reduce the complexity of the system. However, our results suggest that complex, nonlinear dynamics and a high dimensional state space can be beneficial. As we have demonstrate such mechanical structures can be exploited instead of counteracted by a high-level controller. This enormously reduces the control effort and, therefore, increases the efficiency. In addition, because the "computations" performed by the body deliver information about the dynamics of the organism to the organism itself, they can be used to learn what can be called a "body schema", see Hoffmann et al. (2010).

Another remarkable property of our proposed approach is that we do not need to understand the details of the physical dynamics of the system. Actually, we don't even have to know the physical meanings of the readouts. Naturally, this facilitates a learning-based implementation of the approach in real-world applications. Another fact, which promotes real-world implementations, is that noise can be tolerated (and is in fact needed for robust learning), as we have demonstrated. An essential ingredient of our proposed scheme for morphological computation is the availability of a fairly larger number of diverse signals about the current state of the body. These signals are not required to measure well defined physical quantities with high precision. Rather, it suffices if different current states of a complex compliant body cause different high dimensional signals. Hence, for example, a large number of randomly distributed cheap pressure sensors in compliant body parts are likely to suffice. Possibly various time-delayed and mixed versions of such signals could provide even better performance. Thus, our theory for morphological computation provides a novel challenge for the design of sensors in compliant robot systems (since they become an integral part of the morphological computing system). It should be noted on this side that biological organisms receive in general very high dimensional readouts of their body state through a large number sensors [e.g., humans have about 10^7 sensory neurons, see Martini et al. (2011)].

We have considered in this article for the sake of simplicity just generic models of compliant robot parts: random systems of masses and springs. These generic systems, which had not been designed for any particular range of morphological computing tasks, exhibited already remarkable performance (after training of linear feedbacks and readouts). Hence, it is tempting to explore in further research more specialized designs of compliant robot parts that facilitate fast learning of a particular class of morphological computations, e.g., generation of periodic patterns that are needed for locomotion of a particular robot. Such research promises not only to throw new light on our understanding of the design of concrete body parts of biological organisms, but also to provide new ideas for the design of compliant robots. We hope that the theoretical framework of this article will provide principles and concepts that support such further research.

Acknowledgments Written under partial support by the European Union projects project # FP7-231267 (ORGANIC), # 216886 (PASCAL2), # 248311 (AMARSi), and by the Austrian Science Fund FWF, project # P17229-N04. We also want to thank the anonymous reviewers for their very helpful suggestions and comments, Stefan Häusler for fruitful discussions, and Rodney Douglas for his advice regarding biological data.

Appendix

A: Kernel in the context of machine learning

A kernel (in the sense of machine learning, see Vapnik (1998)) 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 $Q(\mathbf{u})$.

B: Definition of feedback linearizable systems

A dynamic system can be shown to be feedback linerizable by applying the following theorem taken from Sontag (1998):

Theorem 1 *A system of the form*

$$\mathbf{x}'(t) = f(\mathbf{x}(t)) + g(\mathbf{x}(t)) \cdot v(t), \tag{8}$$

with $\mathbf{x} = [x_1, \ldots, x_n]^T$, $f : \mathbb{R}^n \to \mathbb{R}^n$ *and* $g : \mathbb{R}^n \to \mathbb{R}^n$ *is feedback linearizable about some point* $\mathbf{x}^0$ *if and only if*

(LI) *The set of vector fields* $\{\mathbf{g}(\mathbf{x}), \mathrm{ad}_\mathbf{f}\mathbf{g}(\mathbf{x}), \ldots, \mathrm{ad}_\mathbf{f}^{n-1}\mathbf{g}(\mathbf{x})\}$ *is linearly independent*

(INV) *The distribution generated by* $\{\mathbf{g}(\mathbf{x}), \mathrm{ad}_\mathbf{f}\mathbf{g}(\mathbf{x}), \ldots, \mathrm{ad}_\mathbf{f}^{n-2}\mathbf{g}(\mathbf{x})\}$ *is involutive*

in some neighborhood of $\mathbf{x}^0$ *(for proof we refer to Sontag (1998)).*

The expression $ad_\mathbf{f}^i\mathbf{g}(\mathbf{x})$ is the i-times recursively applied Lie bracket of f and g, see, for example, Isidori (2001). Accordingly, a dynamic system of the form of Eq. 8 is *globally feedback linearizable*, if and only if the conditions **(LI)** and **(INV)** hold for the whole state space.

C: Restating the orginal theorem by Maass et al. (2007)

For the sake of completeness, we state here again the original basic theorem of Maass et al. (2007) in reference to our notation as discussed in Sect. 2. Note that they used this theorem to proof that generic models of neural networks belong to the class of feedback linearizeable systems and can be, therefore, employed as a fixed computational module $\mathcal{C}$, if they received the appropriate feedback and have the corresponding readout (compare to Fig. 1).

Theorem 2 *A large class* S_n *of systems of differential equations of the form of Eq. 8 are in the following sense universal for analog computing:*

This system 8 can respond to an external input $u(t)$ *with the dynamics of any differential equation of order n of the form of Eq. 3, i.e.,* $\mathcal{G}$ *(for arbitrary smooth functions* $G : \mathbb{R}^n \to \mathbb{R}$*) if the input term* $v(t)$ *in Eq. 8 is replaced by a suitable memoryless feedback function* $K(x_1(t), \ldots, x_n(t), u(t))$*, and if a suitable memoryless readout function* $h(x(t))$ *is applied to its internal state* $\mathbf{x}(t) = \langle x_1(t), \ldots, x_n(t)\rangle$*: one can achieve then that* $h(x(t)) = z(t)$ *for any solution* $z(t)$ *of Eq. 3.*

Also the dynamic responses of all systems consisting of several higher order differential equations of the form Eq. 3 can be simulated by fixed systems of the form of Eq. 8 with a corresponding number of feedbacks.

D: Proof that a nonlinear mass–spring system belongs the class S_n of feedback linearizeable systems

In this section, we demonstrate that a physically realistic, nonlinear mass–spring systems belongs to the class S_n of feedback linearizeable systems. Thus, they can be used as basic systems $\mathcal{C}$ to emulate arbitrary, nonlinear, dynamical systems $\mathcal{G}$ of the form of Eq. 3. Moreover, since the equations describe real physical systems, for example, compliant body parts of the robot, we can conclude that such real physical system can be employed for morphological computation too.

For the proof we have to demonstrate that the dynamic system of Eq. 4, which describes such nonlinear mass–spring systems, belongs to the class S_n of feedback linearizable systems. Accordingly to Theorem 1 the conditions **LI** and **INV** have to be fulfilled, i.e., the set of vector fields $\{\mathbf{g}(\mathbf{x}), \mathrm{ad}_\mathbf{f}\mathbf{g}(\mathbf{x}), \ldots, \mathrm{ad}_\mathbf{f}^{n-1}\mathbf{g}(\mathbf{x})\}$ has to be linearly independent, and the distribution generated by $\{\mathbf{g}(\mathbf{x}), \mathrm{ad}_\mathbf{f}\mathbf{g}(\mathbf{x}), \ldots, \mathrm{ad}_\mathbf{f}^{n-2}\mathbf{g}(\mathbf{x})\}$ has to be involutive.

For the case of the nonlinear mass–spring system (Eq. 4) the order is $n = 2$ and the regarding vector fields are

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} x_2 \\ -p(x_1) - q(x_2) \end{pmatrix} \quad \text{and} \quad \mathbf{g}(\mathbf{x}) = \begin{pmatrix} 0 \\ \frac{1}{m} \end{pmatrix}.$$

The resulting Lie bracket of $\mathbf{f}(\mathbf{x})$ and $\mathbf{g}(\mathbf{x})$ is (now dropping for the sake of readability the reference to the state vector $\mathbf{x}$)

$$ad_\mathbf{f}\mathbf{g} = \underbrace{\nabla\mathbf{g} \cdot \mathbf{f}}_{=0} - \nabla\mathbf{f} \cdot \mathbf{g} = -\begin{pmatrix} 0 & 1 \\ -p' & -q' \end{pmatrix} \cdot \begin{pmatrix} 0 \\ \frac{1}{m} \end{pmatrix},$$
$$= \begin{pmatrix} \frac{1}{m} \\ -\frac{1}{m}q' \end{pmatrix}$$

where $(.)'$ denotes the first derivative in time of the regarding state variable. First, we have to show that following set of vector fields

$$[\mathbf{g}, \mathrm{ad}_\mathbf{f}\mathbf{g}] = \begin{pmatrix} 0 & \frac{1}{m} \\ \frac{1}{m} & -\frac{1}{m}q' \end{pmatrix}$$

is linearly independent (condition **LI**), which is true for any value of q', assuming that the mass $m \neq 0$.

Second, we have to show, that $\mathbf{g} = [0, \frac{1}{m}]^T$ is involutive, which is also true, since it is a constant vector. Hence, the nonlinear mass–spring system of Eq. 4 belongs to the class S_n of feedback linearizable systems and, therefore, can be employed as a basic computation module $\mathcal{C}$ for morphological computation (e.g., the red box in Fig. 1).

Note that theoretically a *linear* mass–spring systems can also be employed. The proof is trivial. The condition **LI** corresponds in the linear case (with $\mathbf{A}$ being the state matrix and $\mathbf{b}$ the input vector of the state space representation of the system) to the condition that the controllabilty matrix $\mathbf{R} = \begin{pmatrix}\mathbf{b}\ \mathbf{Ab}\ \ldots\ \mathbf{A}^{n-1}\mathbf{b}\end{pmatrix}$ is invertible, i.e., the system is controllable. This is true for any physically realistic linear mass–spring system (i.e., the mass is not zero and the damping factor is positive). The second condition is true for any linear system (see Theorem 6.2 in Slotine and Li (1991)). Although theoretically one could use linear systems, in the case of having linear feedbacks and linear readouts it is beneficial that the physical body consists of nonlinear systems in order to be able to exhibit the *kernel* property, as discussed in Sect. 4.

E: Proof that an array of linear mass–spring systems belongs the class S_n of feedback linearizeable systems

Now we consider a set of **linear** mass–spring systems (along with the proof for neural networks equations like (11) in Maass et al. (2007), where they used linear systems in parallel too). Assuming our basic module has the following form

$$\begin{aligned} x_1' &= x_2 \\ x_2' &= -kx_1 - dx_2 + v, \end{aligned} \tag{9}$$

where $k \in \mathbb{R}^+$ is the linear spring constant and $d \in \mathbb{R}^+$ the linear damping constant. The same system can be written in matrix form

$$\begin{pmatrix} x_1' \\ x_2' \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & 1 \\ -k & -d \end{pmatrix}}_{\mathbf{A}_1} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \underbrace{\begin{pmatrix} 0 \\ 1 \end{pmatrix}}_{\mathbf{b}_1} v$$

$$y = \underbrace{\begin{pmatrix} 1\ 0 \end{pmatrix}}_{\mathbf{c}_1^T} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}.$$

We now consider a system, which is made of m different parallel linear mass–spring systems of the form of Eq. 9, with $x_{i,j}$ being the jth state variable of the ith system ($j = 1, 2$ and $i = 1, 2, \ldots, m$).

$$\begin{pmatrix} x_{1,1}' \\ x_{1,2}' \\ x_{2,1}' \\ x_{2,2}' \\ \vdots \\ x_{m,1}' \\ x_{m,2}' \end{pmatrix} = \underbrace{\begin{pmatrix} \mathbf{A}_1 & 0 & \cdots & 0 \\ 0 & \mathbf{A}_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{A}_m \end{pmatrix}}_{\mathbf{A}} \begin{pmatrix} x_{1,1} \\ x_{1,2} \\ x_{2,1} \\ x_{2,2} \\ \vdots \\ x_{m,1} \\ x_{m,2} \end{pmatrix} + \underbrace{\begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ \vdots \\ 0 \\ 1 \end{pmatrix}}_{\mathbf{b}} v \tag{10}$$

This system of m sub-systems in parallel has the order of $n = 2m$. Now we have show that it fulfills the two conditions **LI** and **INV** (see Theorem 1). Note that a linear system trivially fulfills the second condition **INV** (see Theorem 6.2 in Slotine and Li (1991)) and the first condition **LI** takes a special form. The corresponding vector fields are $\mathbf{f} = \mathbf{A}$ and $\mathbf{g} = \mathbf{b}$, and the set of vector fields $\{\mathbf{g}(\mathbf{x}), ad_{\mathbf{f}}\mathbf{g}(\mathbf{x}), \ldots, ad_{\mathbf{f}}^{n-1}\mathbf{g}(\mathbf{x})\}$ becomes therefore $\begin{pmatrix}\mathbf{b}\ \mathbf{Ab}\ \ldots\ \mathbf{A}^{n-1}\mathbf{b}\end{pmatrix}$. In control theory this matrix is well known as the so-called controllability matrix $\mathbf{R}$, see, e.g., Slotine and Li (1991). In order to demonstrate that condition **LI** is fulfilled we have to show, that $\mathbf{R} = \begin{pmatrix}\mathbf{b}\ \mathbf{Ab}\ \ldots\ \mathbf{A}^{n-1}\mathbf{b}\end{pmatrix}$ is invertible. Our proof is based on following observations: The sub-systems are **non** interacting (parallel systems), i.e., no state variable from the kth system has influence on any state variable of the lth system with $k \neq l\ \forall k, l = 1, 2, \ldots, n$ at any time. Therefore, $\mathbf{R}$ evolves in such a way that the two corresponding rows of the ith system only depend on its own system variables k_i and d_i. For example row 1 and 2 of $\mathbf{R}$ only depend on k_1 and d_1, row 3 and 4 only on the 2nd subsystem, and so on. Any pair of such rows of $\mathbf{R}$ have the following form for a given order n

$$\begin{pmatrix} 0 & 1 & d_i & p(d_i^2) & \cdots & p(d_i^{(n-2)}) \\ 1 & d_i & p(d_i^2) & p(d_i^3) & \cdots & p(d_i^{(n-1)}) \end{pmatrix}, \tag{11}$$

where $p(d_i^w)$ denotes a polynomial of d_i of order w. Note that any $p(d_i^w)$ also depends on k_i in some polynomial form with an order lower than w, but for the sake of readability it has been suppressed here. The proof holds independently from that for any real positives values of k_i. Assuming we have different sub-systems (i.e., $k_i \neq k_j$ and $d_i \neq d_j$ for $i \neq j \forall i, j = 1, 2, \ldots, m$) it is easy to see from the structure above, that all columns and all rows a linearly independent, hence, the matrix is invertible.

Therefore, we have shown that any system of the form of Eq. 10, with different sub-systems (as defined above), has a controllability matrix $\mathbf{R}$, which is invertible. Hence, the overall system fulfills both conditions **LI** and **INV** and, therefore, belongs to the class of feedback linearizable system S_n.

References

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

Bishop CM (1994) Training with noise is equivalent to tikhonov regularization. Neural Comput 7:108–116

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

Franceschini N, Pichon JM, Blanes C, Brady JM (1992) From insect vision to robot vision. Phil Trans R Soc Lond B 337(1281):283–294

Hauser H, Ijspeert A, Füchslin RM, Pfeifer R, Maass W (2012) Towards a theoretical foundation for morphological computation with compliant bodies. Biol Cybern 105(5):355–370. ISSN 0340-1200

Hoerzer G, Legenstein R, Maass W (2011) Eliminating the teacher in reservoir computing. In: Pfeifer R, Sumioka H, Füchslin RM,

Hauser H, Nakajima K, Miyashita S (eds) Proceedings of the 2nd international conference on morphological computation, Venice
Hoerzer GM, Legenstein R, Maass W (2012) Emergence of complex computational structures from chaotic neural networks through reward-modulated hebbian learning (in preparation)
Hoffmann M, Marques H, Hernandez Arieta A, Sumioka H, Lungarella M, Pfeifer R (2010) Body schema in robotics: a review. IEEE Trans Auton Mental Develop 2(4):304–324
Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural Networks 2:359–366
Iida F, Pfeifer R (2006) Sensing through body dynamics. Robot Auton Syst 54(8):631–640
Ijspeert A, Crespi A, Ryczko D, Cabelguen J-M (2007) From swimming to walking with a salamander robot driven by a spinal cord model. Science 315(5817):1416–1420
Isidori A (2001) Nonlinear control systems, 3rd edn. Springer, London
Khalil HK (2002) Nonlinear systems, 3rd edn. Prentice Hall, Englewood Cliffs
Legenstein R, Chase SM, Schwartz AB, Maass W (2010) A reward-modulated Hebbian learning rule can explain experimentally observed network reorganization in a brain control task. J Neurosci 30(25):8400–8410
Lukoševičius M, Jaeger H (2009) Reservoir computing approaches to recurrent neural network training. Comput Sci Rev 3(3):127–149
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
Maass W, Joshi P, Sontag ED (2007) Computational aspects of feedback in neural circuits. PLoS Comput Biol 3(1):e165
Martini HF, Nath JL, Bartholomew EF (2011) Fundamentals of anatomy & physiology, 9th edn. Benjamin-Cummings Publishing Company, Menlo Park
McGeer T (1990) Passive dynamic walking. Int J Rob Res 9(2):62–82
Miyashita S, Göldi M, Pfeifer R (2011) How reverse reactions influence the yield rate of stochastic self-assembly. Int J Robot Res 30:627–641
Palm WJ III (1999) Modeling, analysis, and control of dynamic systems, 2nd edn. Wiley, New York
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
Pfeifer R, Bongard JC (2007) How the body shapes the way we think. The MIT Press, Cambridge
Pfeifer R, Lungarella M, Iida F (2007) Self-organization, embodiment, and biologically inspired robotics. Science 318:1088–1093
Rieffel J, Trimmer B,Lipson H (2008) Mechanism as mind: what tensegrities and caterpillars can teach us about soft robotics. In: Bullock S, Noble J, Watson R, Bedau MA (eds) Artificial life XI: proceedings of the eleventh international conference on the simulation and synthesis of living systems, pp 506–512. MIT Press, Cambridge
Righetti L, Ijspeert AJ (2008) Pattern generators with sensory feedback for the control of quadruped locomotion. In: IEEE international conference on robotics and automation, pp 819–824. ICRA 2008, Pasadena, 19–23 May 2008
Schrauwen B, Verstraeten D, Van Campenhout J (2007) An overview of reservoir computing: theory, applications and implementations. In: Proceedings of the 15th European symposium on artificial neural networks pp 471–482. ESANN , Bruges
Shaw RS, Packard N, Schröter M, Swinney HL (2007) Geometry-induced asymmetric diffusion. PNAS 104(23):9580–9584
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, pp 756–765. Springer, Berlin
Slotine J-JE, Li W (1991) Applied nonlinear control, 1st edn. Prentice Hall, New York
Sontag ED (1998) Mathematical control theory: deterministic finite dimensional systems, 2nd edn. Springer, New York
Taga G (1998) A model of the neuro-musculo-skeletal system for anticipatory adjustment of human locomotion during obstacle avoidance. Biol Cybern 78(1):9–17
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
Vapnik VN (1998) Statistical learning theory. Wiley, New York
Whitesides GM, Grzybowski B (2002) Self-assembly at all scales. Science 295(5564):2418–2441
Wisse M, van der Linde RQ (2007) Delft pneumatic bipeds, vol 34. Springer Tracts in Advanced Robotics. Springer, Berlin
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), Kyoto
Wood RJ (2007) Design, fabrication, and analysis of a 3DOF, 3 cm flapping-wing MAV, pp 1576–1581. IROS, San Diego
Wyffels F, Schrauwen B (2009) Design of a central pattern generator using reservoir computing for learning human motion. In: ATEQUAL 2009: 2009 ECSIS symposium on advanced technologies for enhanced quality of life (LABRS and ARTIPED 2009): proceedings, pp 118–122. IEEE Computer Society, Los Alamitos
Ziegler M, Iida F, Pfeifer R (2006) "Cheap" underwater locomotion: roles of morphological properties and behavioural diversity. In: International Conference on Climbing and Walking Robots, CLAWAR, Karlsruhe