A Theoretical Framework for Morphological Computation

Wolfgang Maass
Institute for Theoretical Computer Science
Technische Universität Graz, Austria

I will focus on the following aspects of the
Vision of Morphological Computation

- *A body part, especially a compliant body part, may support its control by facilitating computational processes*

- *A complex body part could in addition facilitate learning of motor control*

General characterization of Morphological Computation at ICMC1:

--should have inputs/outputs

--should be programmable

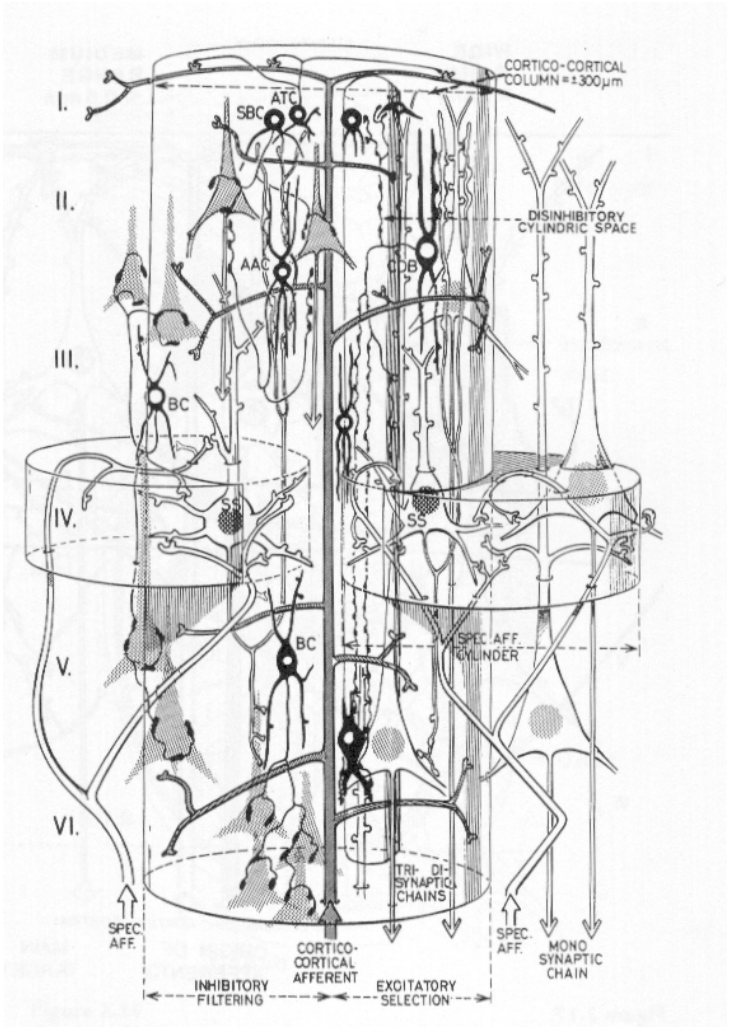
--should have purpose/goal.

We propose a theoretical framework for these aspects of morphological computation, based on ideas from liquid computing (reservoir computing)

joint work with
Helmut Hauser, Auke Ijspeert, Rudolf Füchslin, Rolf Pfeifer

This theoretical framework can explain on the basis of rigorous mathematical results why (under specific conditions) morphological computation can be carried out by systems of masses and springs
(which may be viewed as simple models for complex body parts)

The liquid computing paradigm theory had been invented while trying to solve a somewhat related problem:

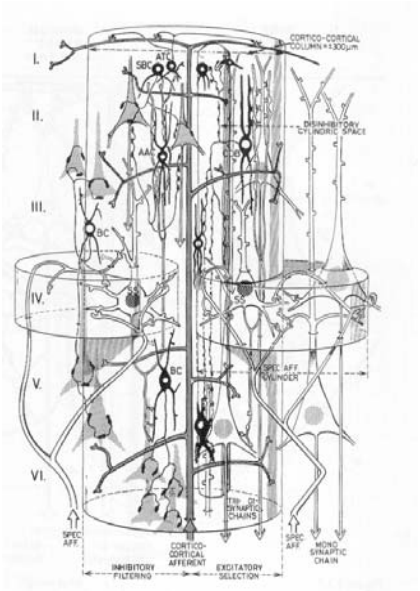


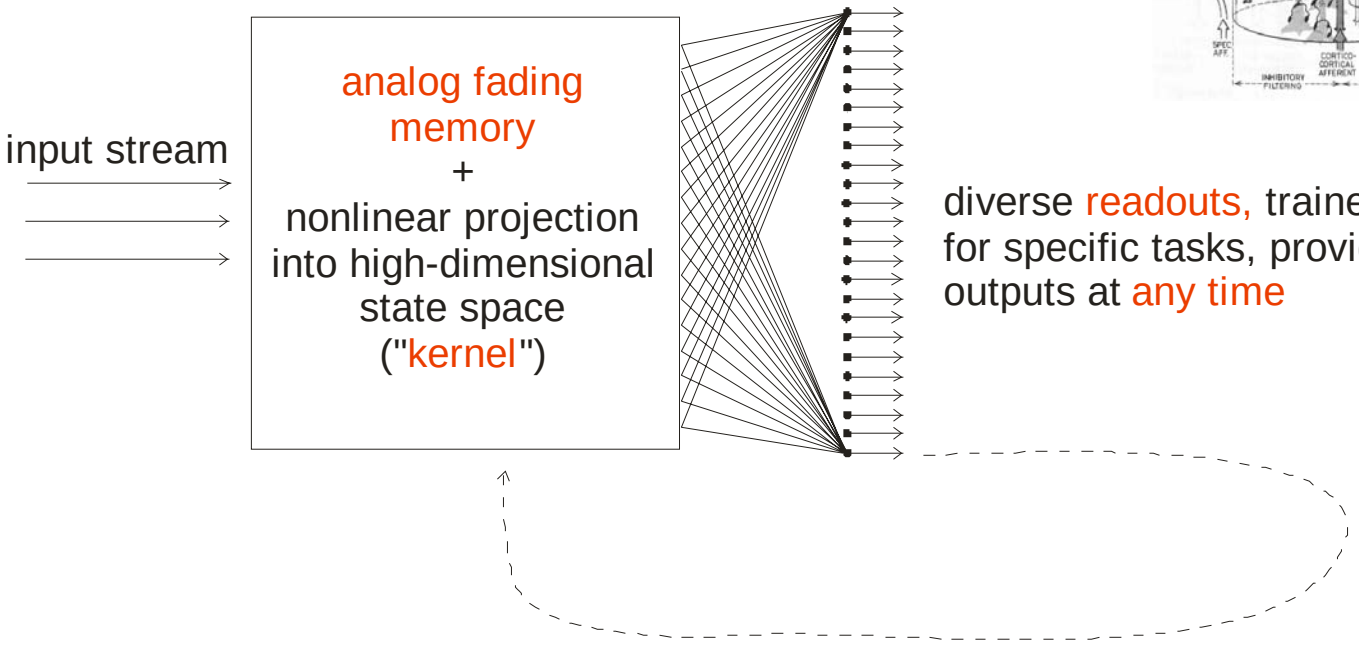
Understand, how a cortical microcircuit, consisting of dozens of different neuron types and synapse types, each with different type of noise, inherent dynamic processes and time constants, could possibly support brain computation and learning.

The traditional model for computing in complex dynamical systems was based on attractors, but these are rarely observed in simultaneous recordings from many neurons.

In addition, the brain needs to carry out *online* computations (or in fact: *anytime computations*).

Resulting computational model for a cortical microcircuit

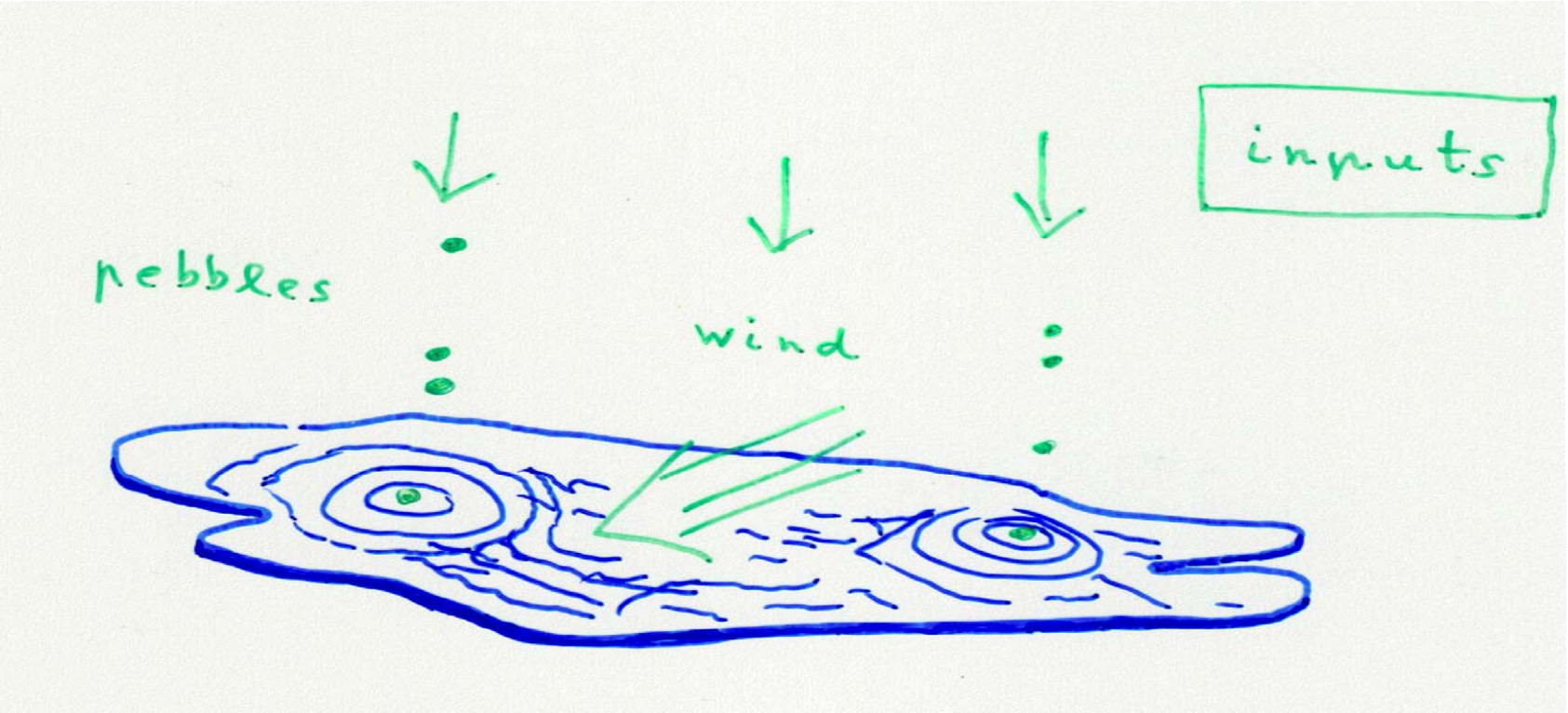




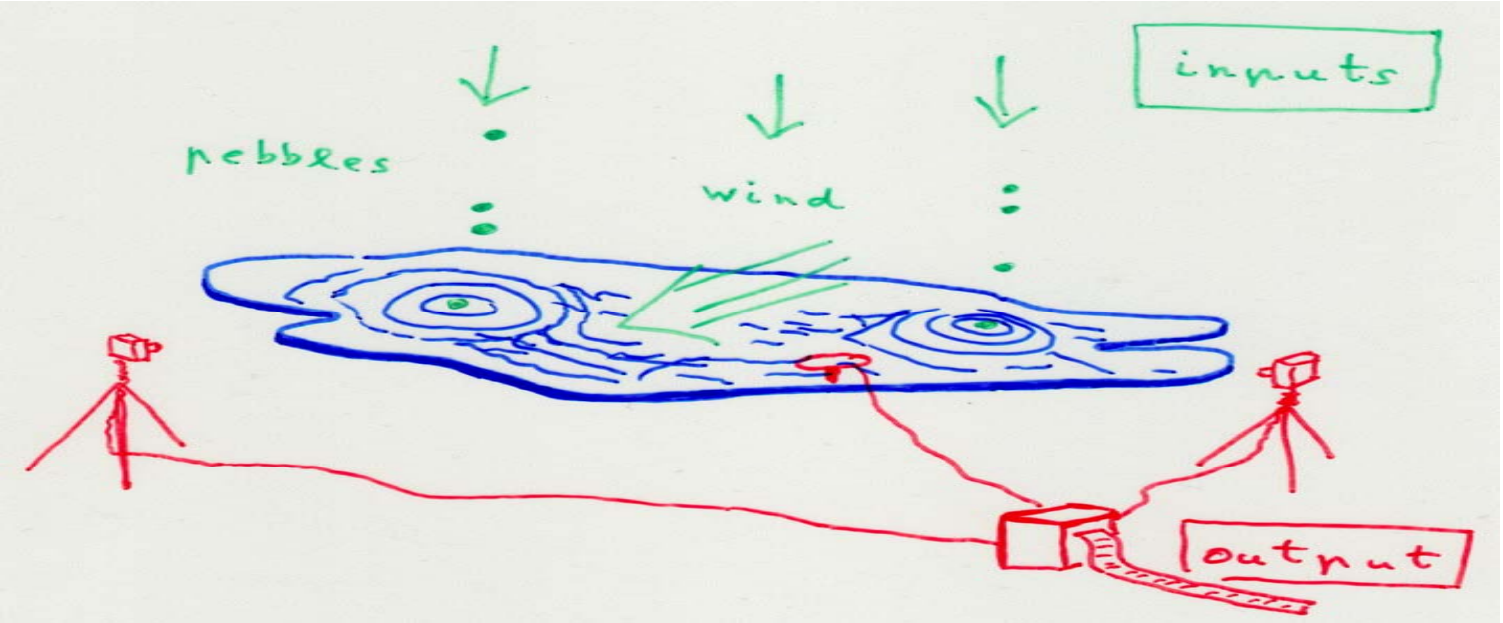
Liquid State Machine [Maass, Natschläger, Markram, 2002]

Why was this called Liquid Computing ?

We wanted to have a model for online computing in dynamical systems.
A pond of water is a nice paradigm for the features that this model emphasizes:



How can a pond of water compute ?

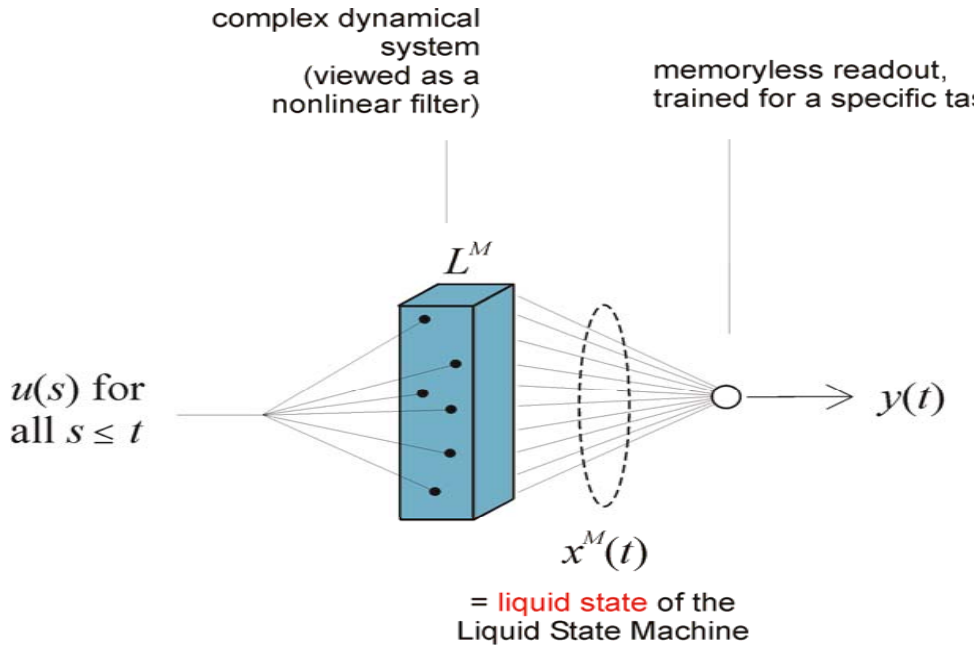


Idea: Add suitable read-out devices, which can be chosen to be memoryless and linear, and which can be trained to extract features that are salient for particular computational tasks.

Note: This computing paradigm does not require the existence of any attractors in the dynamical system (except for the trivial attractor with no activity).

Formal definition of a Liquid State Machine

Note: It generalizes finite state machines to continuous input values u(s), continuous output values y(t), and continuous time t.

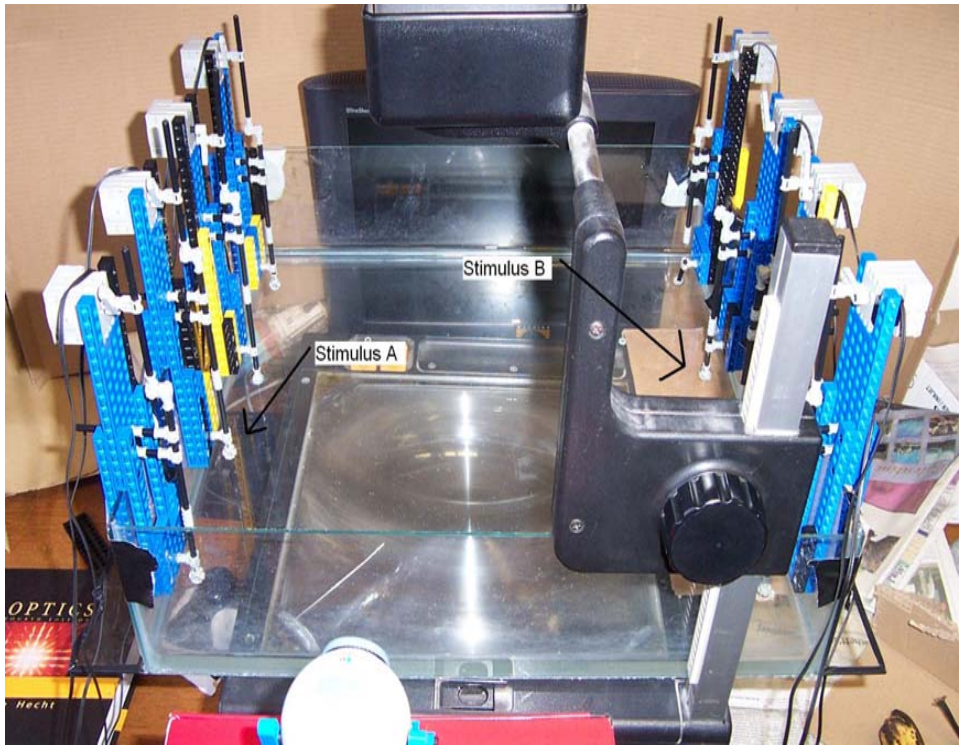


$$x^M(t) = (L^M u)(t)$$
$$y(t) = f^M(x^M(t))$$

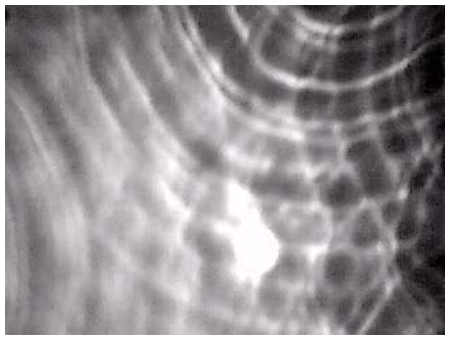
We usually assume that the readout function f is trained for a particular computational task, whereas the dynamical system L is „general purpose“.

A very literal realization of a Liquid State Machine

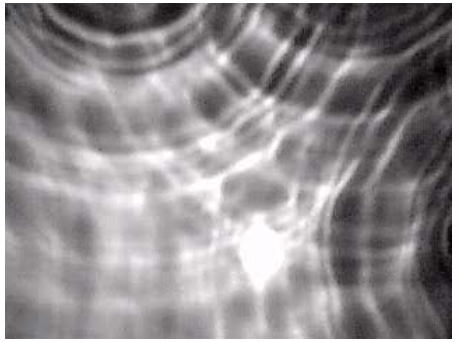
Fernando and Sojakka: Pattern recognition in a bucket:
A real liquid brain, ECAL 2003



„zero“

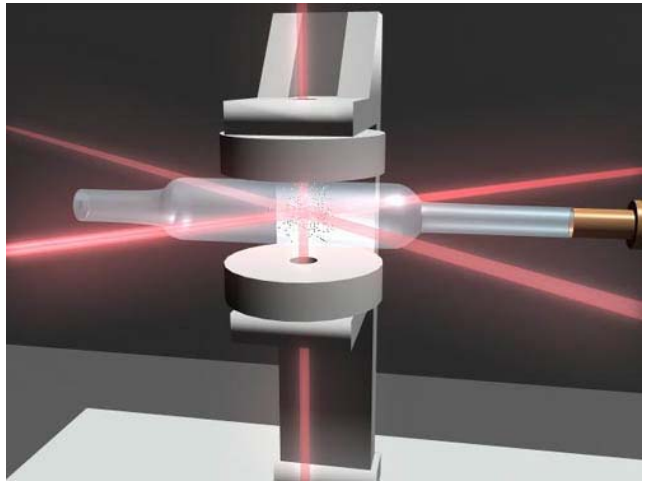


„one“



Some current work on technological applications of liquid computing

- Recognition of continuous speech (Univ. of Gent)
- Reading of handwriting (Int. Univ. Bremen)
- Motor control in robotics (EPFL, U. of Gent)
- Optical computing (Univ. of Gent)



A historical note:

Herbert Jäger had discovered simultaneously related methods for computing and learning with (noise-free) artificial neural networks („echo state networks“). Echo state networks are typically noise-free.

Benjamin Schrauwen coined for the common aspects of both models the name „Reservoir Computing“.

What is the computational power of a Liquid State Machine ?

Their computational power can be characterized in terms of *Volterra Series*:

These are those computational operations (filters) F on input streams u(s) that are time-invariant (i.e, input driven), and only require a fading memory;

$$(F\,u(\cdot))(t) = \alpha_1 \int_0^{\infty} d\,\tau_1\, h_1(\tau_1)\, u(t-\tau_1)$$

$$+ \alpha_2 \int_0^{\infty}\int_0^{\infty} d\,\tau_1\, d\,\tau_2\, h_1(\tau_1,\tau_2)\, u(t-\tau_1)\cdot u(t-\tau_2)$$

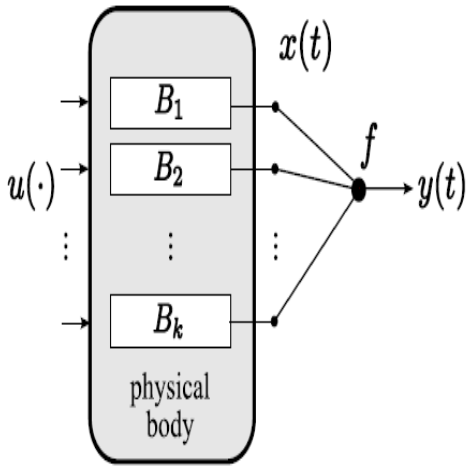
$$+\ldots$$

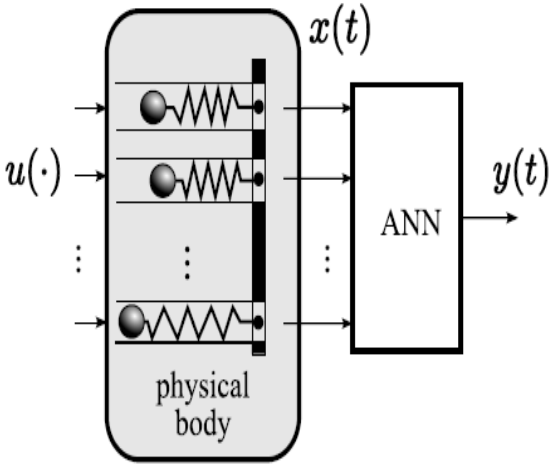
Theorem: (based on [Boyd and Chua, 1985])

Any filter F which is defined by a Volterra series can be approximated with any desired degree of precision by such very simple computational model

- ***if*** there is a rich enough pool **B** of basis filters (time invariant, with fading memory) from which the basis filters $B_1,\ldots,B_k$ in the filterbank can be chosen (**B** needs to have the pointwise separation property) *and*

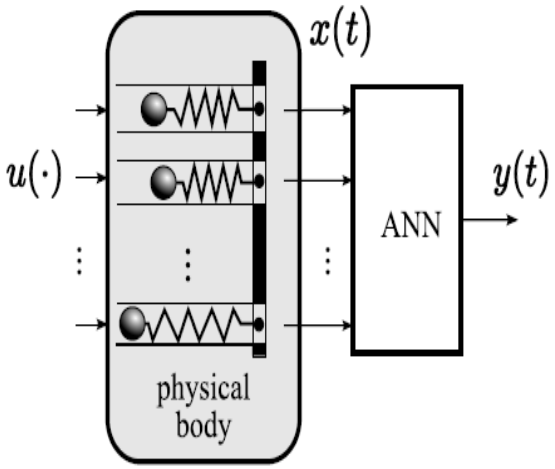
- ***if*** there is a rich enough pool **R** from which the readout functions f can be chosen (**R** needs to have the universal approximation property, i,e. any bounded continuous function can be approximated by function from **R**).

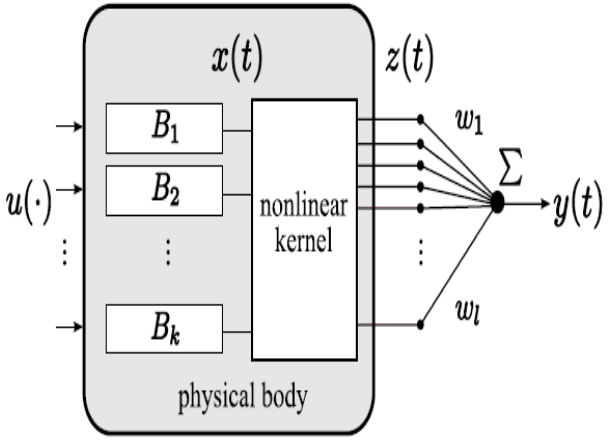


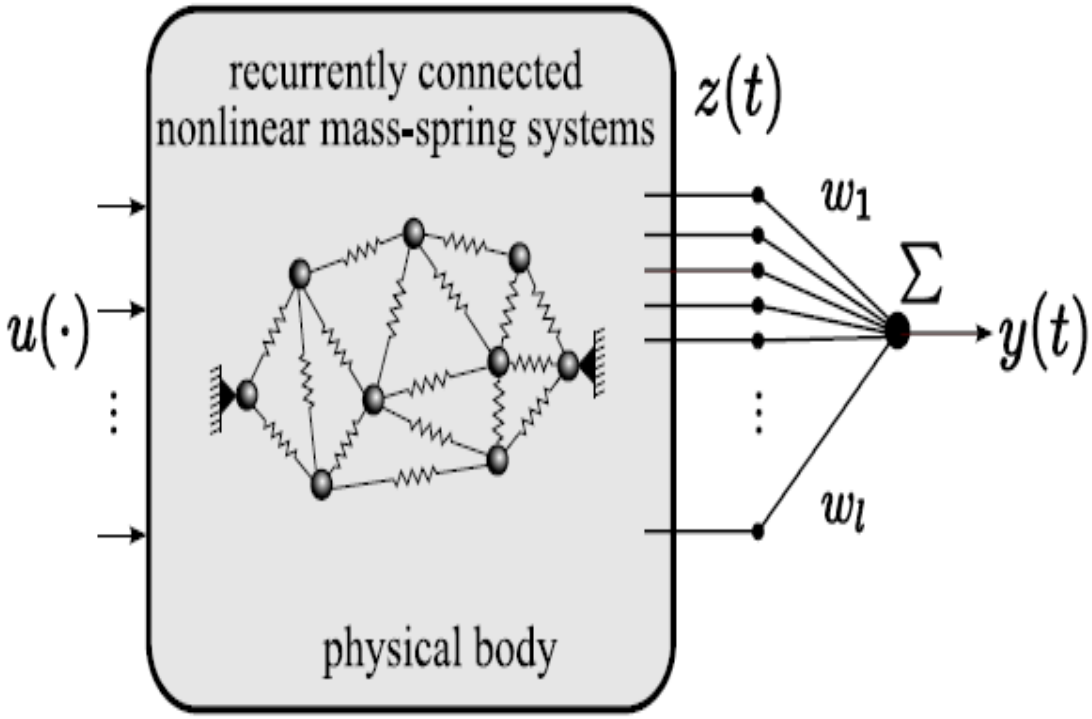


Def: A class **B** of basis filters has the pointwise separation property if there exists for any two input functions $u(\bullet)$, $v(\bullet)$ with $u(s) \neq v(s)$ for some $s \leq t$ a basis filter $B \in$ **B** with $(Bu)(t) \neq (Bv)(t)$.

If one wants to train *linear* readouts for specific tasks, the morphological computation needs to provide in addition nonlinear combinations of internal signals







Nonlinear sprimgs were used in our simulations

What is a kernel
(in the sense of machine learning) ?

A kernel provides numerous nonlinear combinations of input variables, in order to boost the expressive power of any subsequent linear readout.

Example: *If a circuit precomputes all products* $x_i \cdot x_j$ *of n input variables* $x_1,...,x_n$*, then a subsequent linear readout can compute any quadratic function of the original input variables* $x_1,...,x_n$.

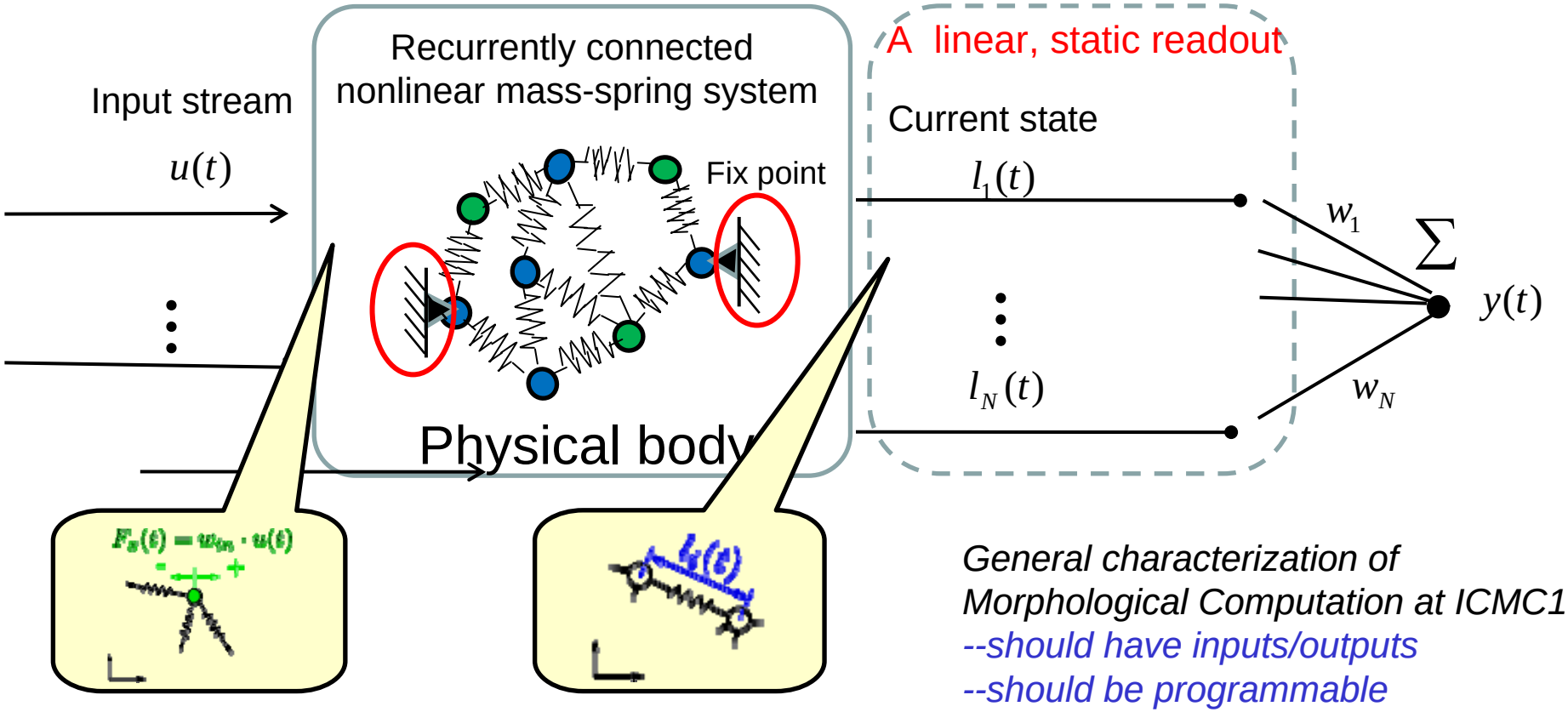
More abstractly, a kernel should map pairwise different input vectors onto linearly independent output vectors (note that this more general goal does not require precise execution of any nonlinear computation).

Remarks:

- Restricting learning to linear readouts has the advantage that learning cannot get stuck in local minima of the error function.
- The same fixed kernel can serve many completely different linear readouts.

Resulting theoretical model for morphological computation of a given filter (time-invariant, with fading memory) (without feedback)

Hypothesis: A recurrent mass-spring system can carry out both the temporal integration and the nonlinear processing of a typical liquid state machine

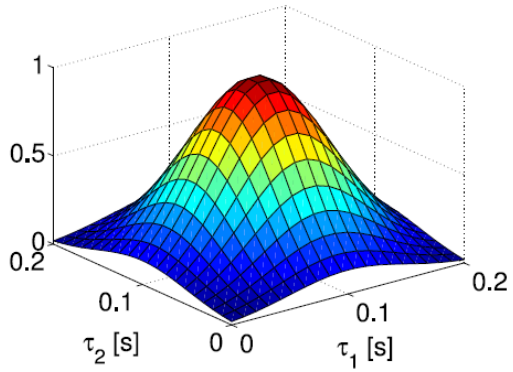


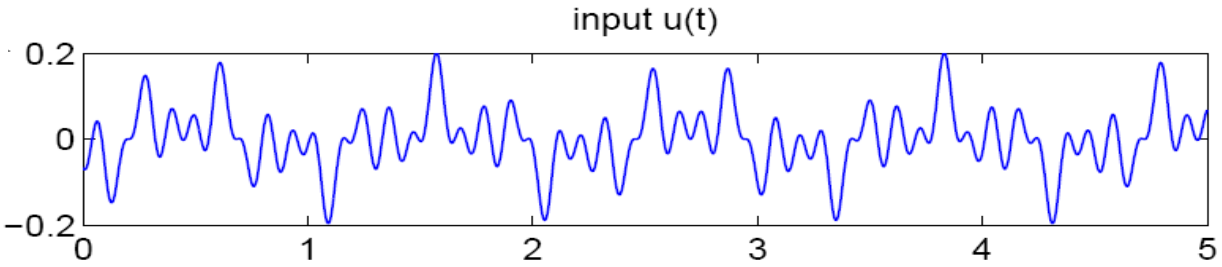
General characterization of Morphological Computation at ICMC1.
--should have inputs/outputs
--should be programmable
--should have purpose/goal.

Other implementations of inputs and outputs also work.

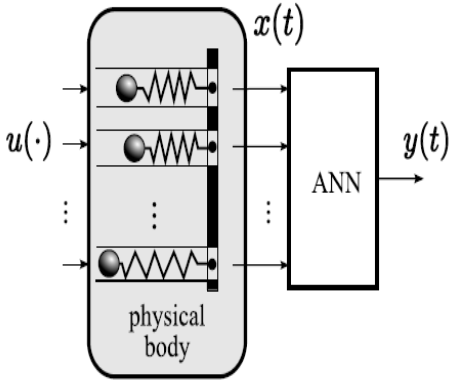
A first test: Computing a 2nd order Volterra filter through morphological computation

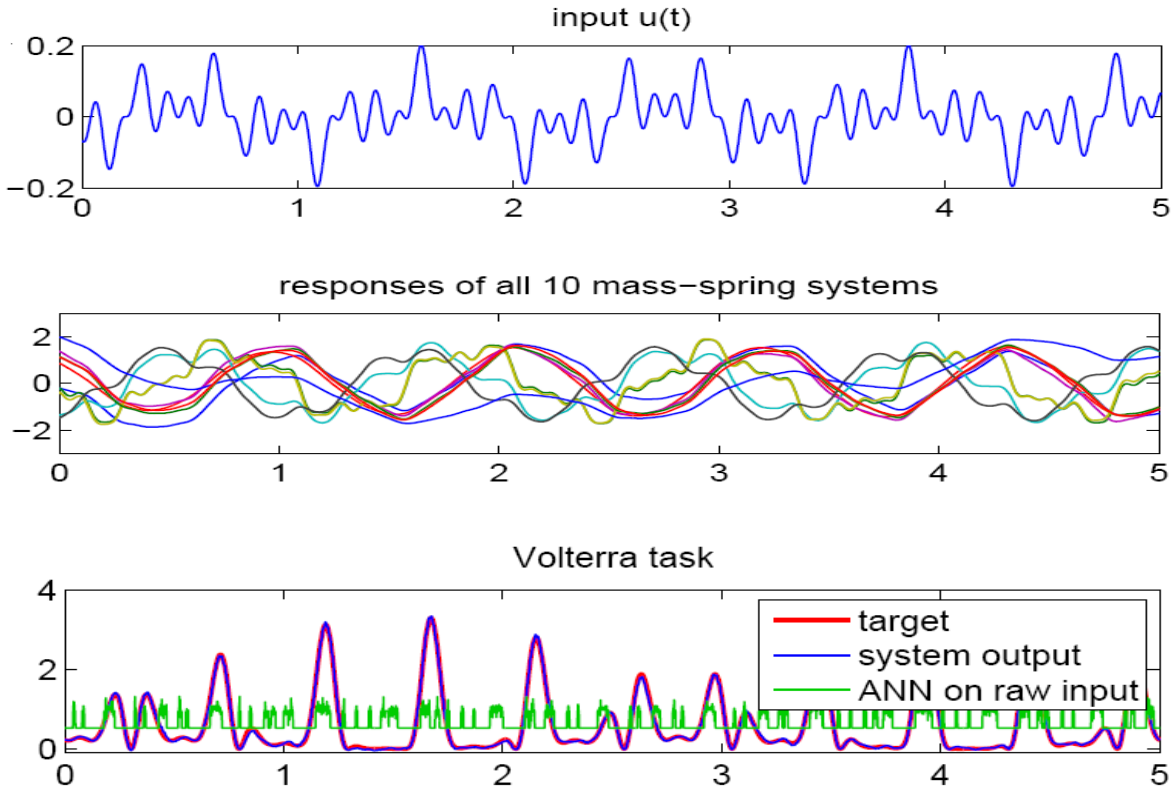
$$y(t) = \mathcal{V}u(t) =$$
$$= \iint_{\tau_1,\tau_2 \in \mathbb{R}_0^+} h_2(\tau_1,\tau_2)u(t-\tau_1)u(t-\tau_2)d\tau_1 d\tau_2$$





Result of this test (for a network consisting of 10 mass spring systems)

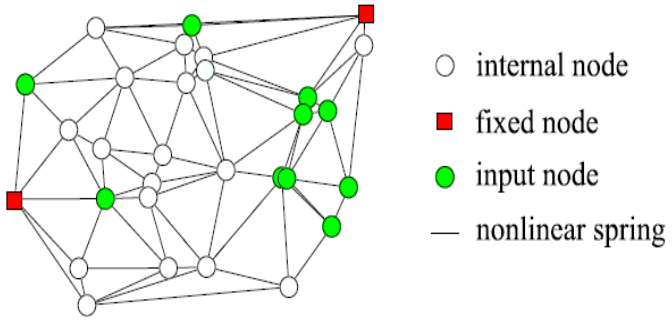


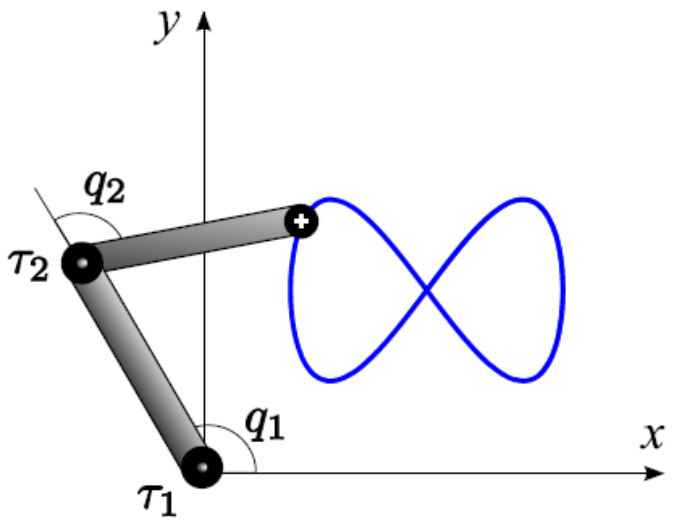


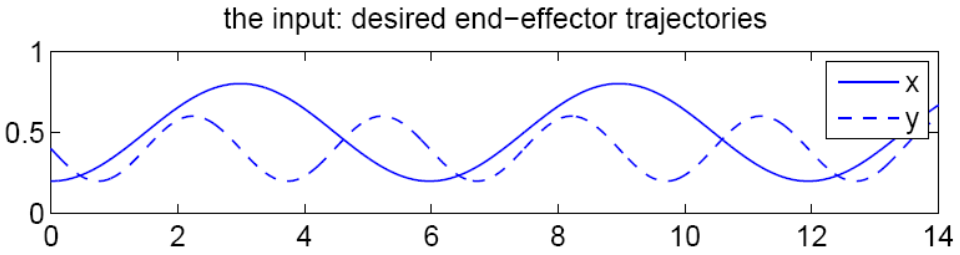
Almost perfect morphological computation of this 2nd order Volterra term (for test inputs that had not been used for training).

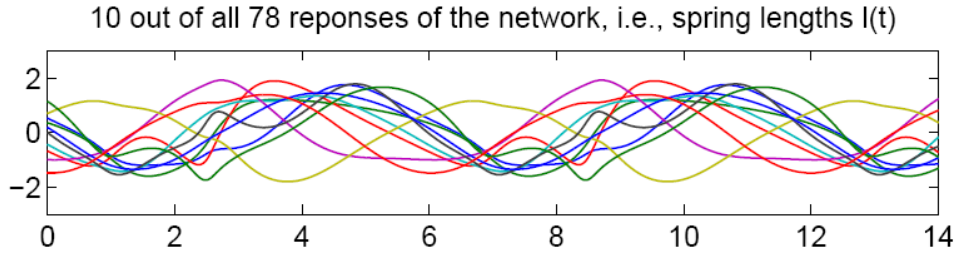
A second test: Morphological computation of the inverse dynamics of a 2D robot arm

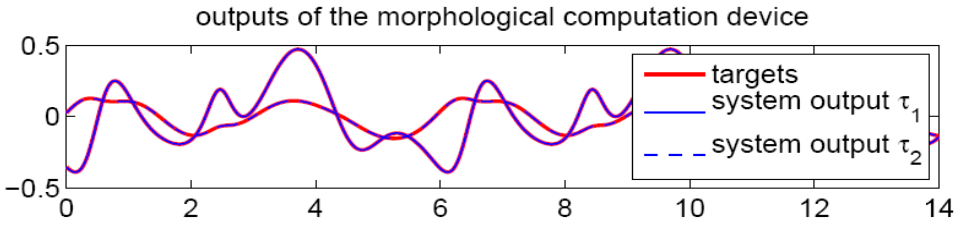
with a "random" network of 30 diverse masses and 78 nonlinear springs, using just linear readouts







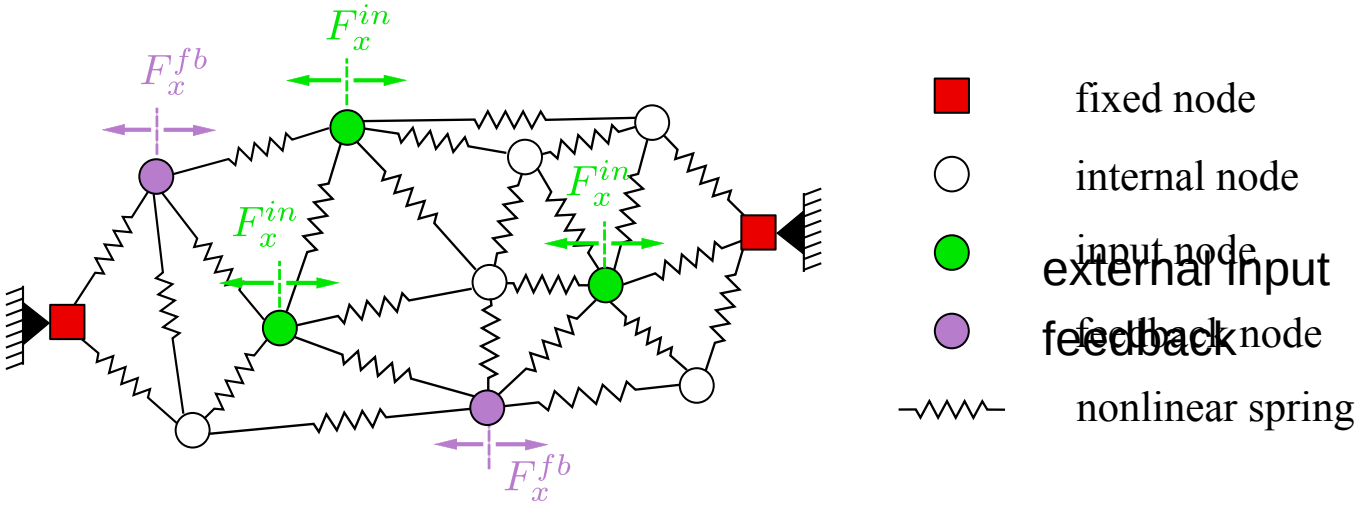




Does the computational power of this system increase if *feedback* from some readout is fed back (as a force) into the mass-spring system ?

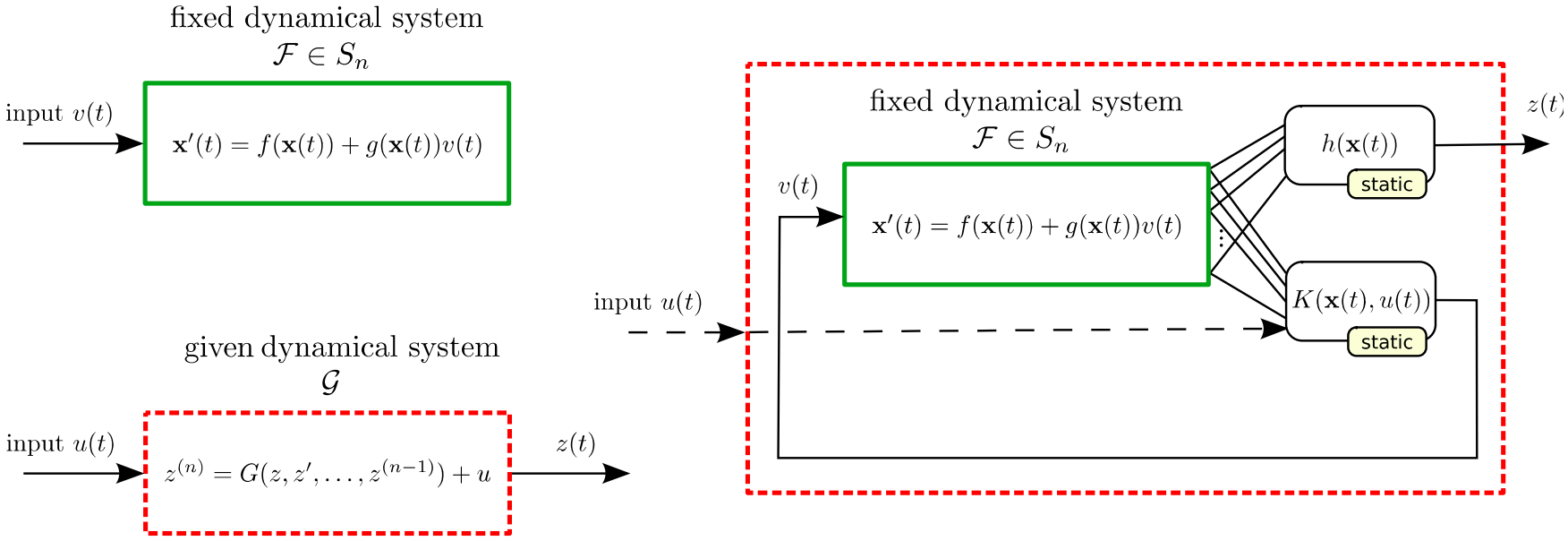
The feedback could for example be implemented as a force on a random set of masses in the system, that is (like all other readouts) a weighted sum of the „liquid state“, in this case of the lengths of the springs in the system.

The weights of this weighted sum are adapted for a given task.



Feedback nodes are randomly chosen. They receive the output in form

Theoretical result: The possible input/output behaviour of the system jumps from Volterra series to quite arbitrary dynamical systems through the addition of feedback (but continuous readout function h and feedback function K should ideally be allowed to be nonlinear)



Any feedback linearizable dynamical system C becomes through the addition of suitable continuous feedback functions K (and readouts h) in this sense *universal for analog computation* [Maass, Joshi, Sontag, PLoS Comp. Biology 2007]

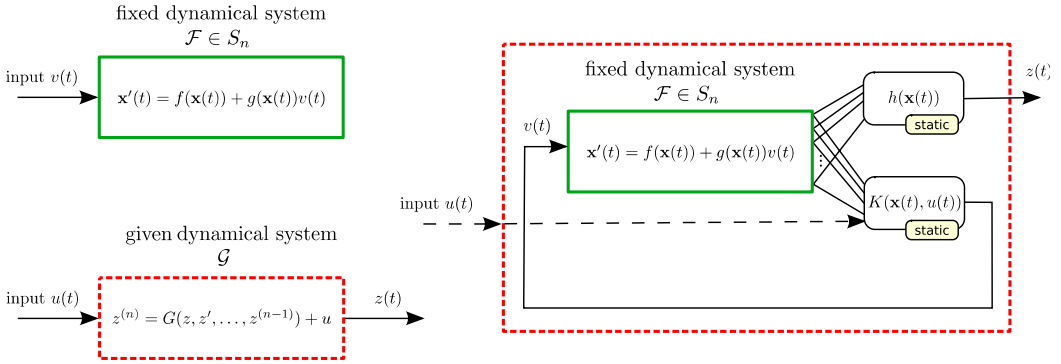
This abstract mathematical result can be applied to mass-spring systems, predicting that they become

universal analog computers

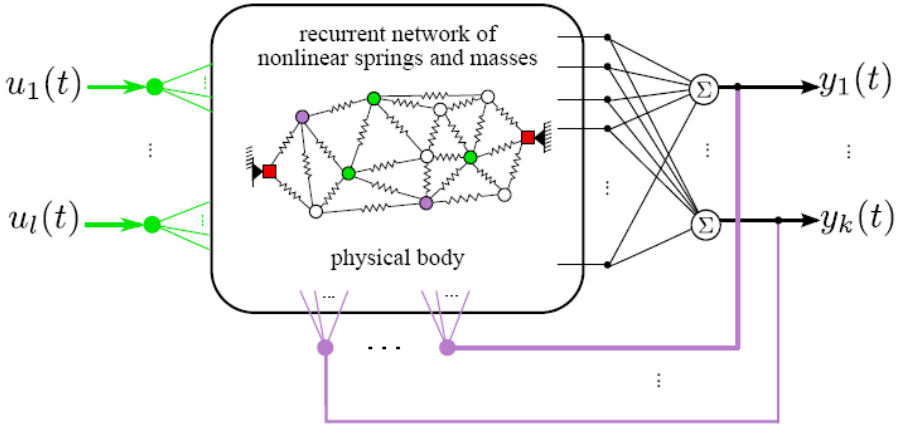
if suitable feedback is added

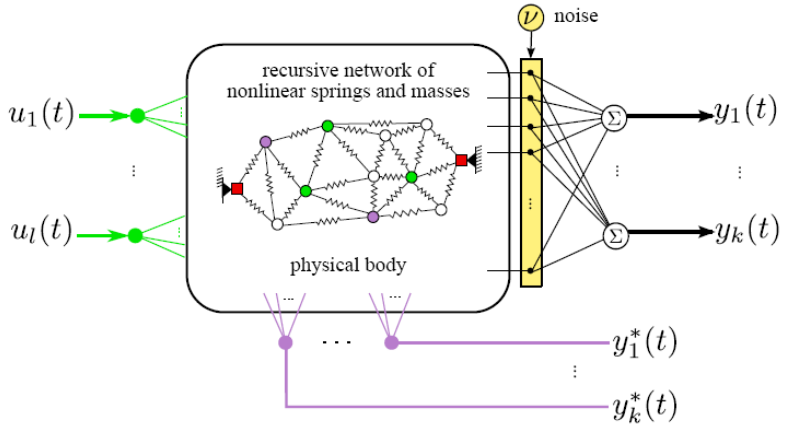
One can prove that standard models for nonlinear springs are feedback linearizable, hence they satisfy the condition of the system C for becoming able to simulate any given dynamical system, if suitable feedback is added.

If one wants to use just linear feedback and readout functions, one needs that the mass-spring system also has the properties of a nonlinear kernel.



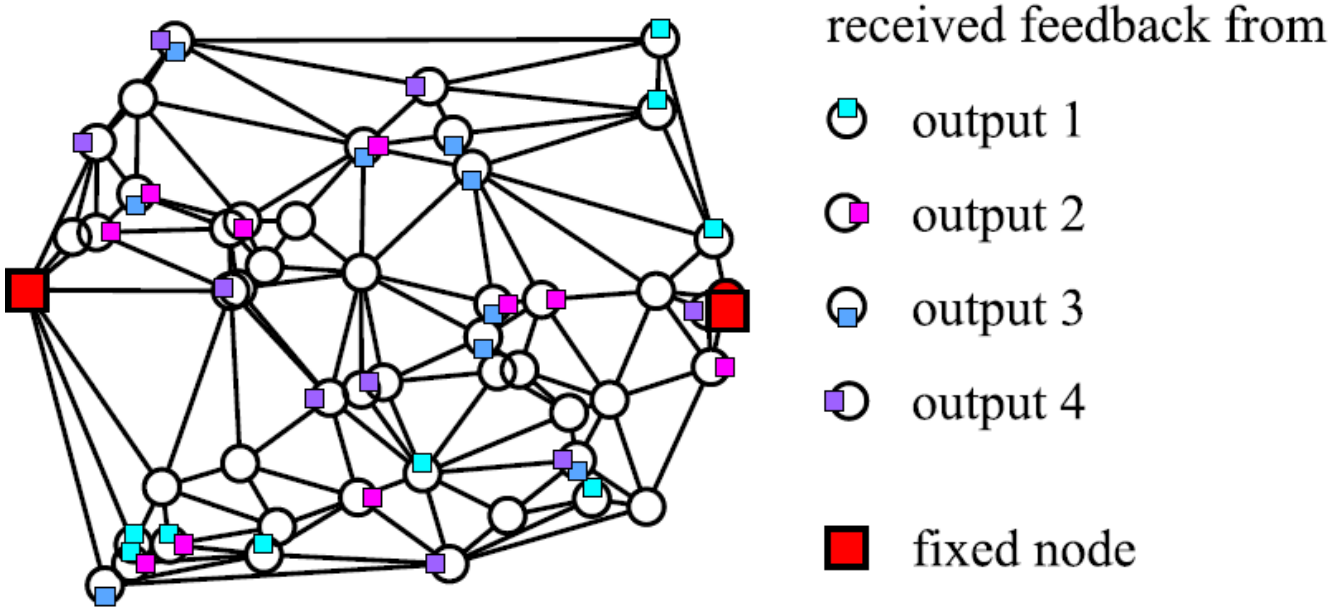
For our tests of morphological computation with feedback, the weights of linear feedbacks K and readouts h were learned (using teacher forcing)





Teacher forcing means that the feedback is replaced by a teacher signal) during training.

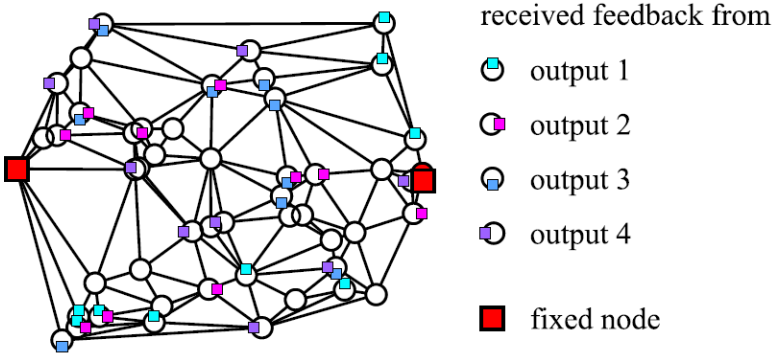
A first test of morphological computation with feedback: Generation of walking patterns for a quadruped robot



Four linear readouts were fed back into the system. The goal was to generate given with these a given 4-dimensional periodic patterns, as for example needed for locomotion.

The target patterns were generated according to the model of [Righetti and Ijspeert, 2008].

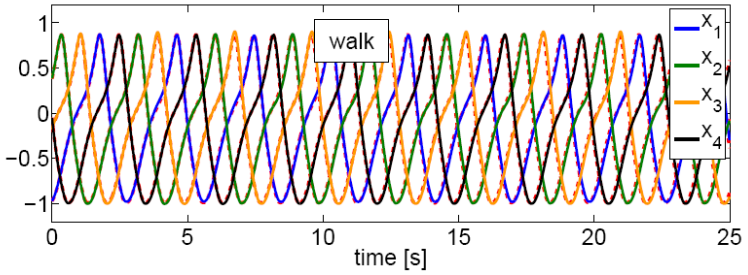
Performance



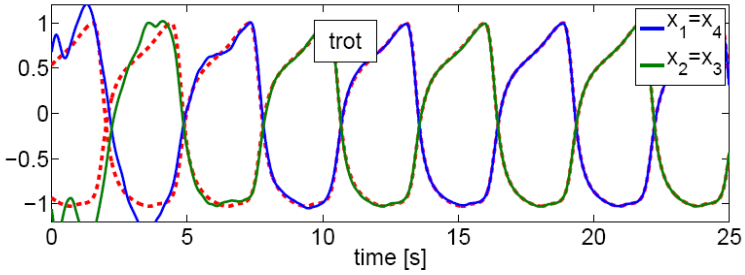
Four different settings of the linear readouts were learned (in the form of four settings of the readout weights) for generating the four different periodic patterns.

The gait is switched by switching the readout weights. The new pattern that is produced converges fast to the new gait..

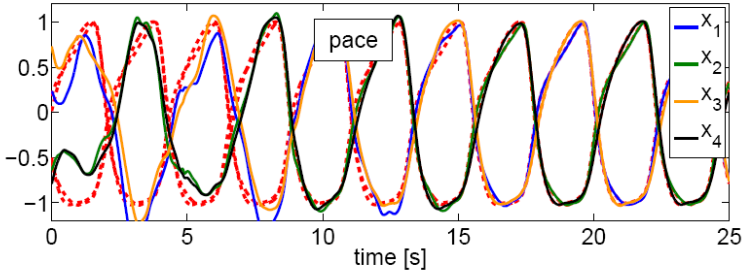
Target outputs are indicated as red dotted lines.



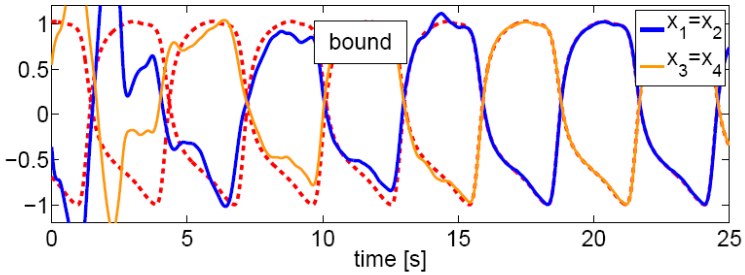
(a)



(b)



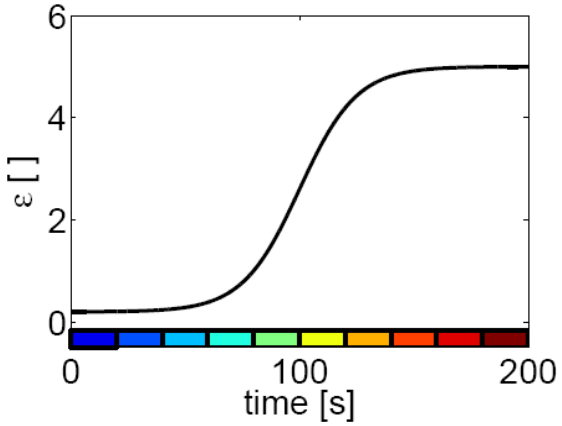
(c)

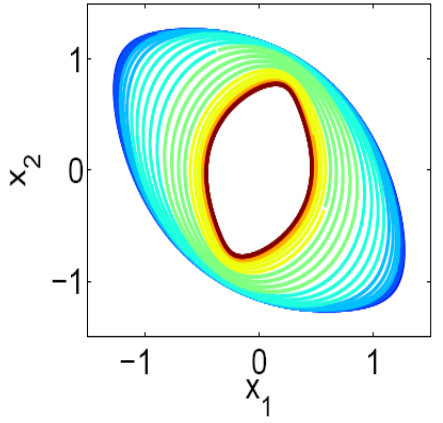


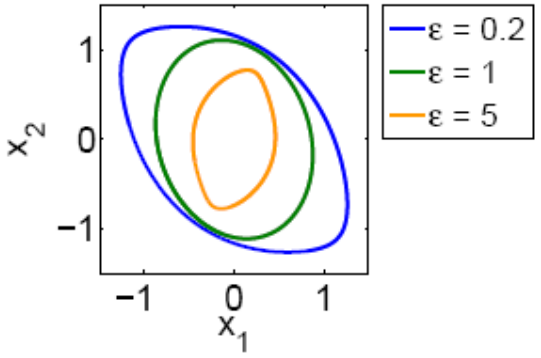
(d)

The morphological computation device can learn to switch on its own the generated pattern, in dependence of the value of an external input

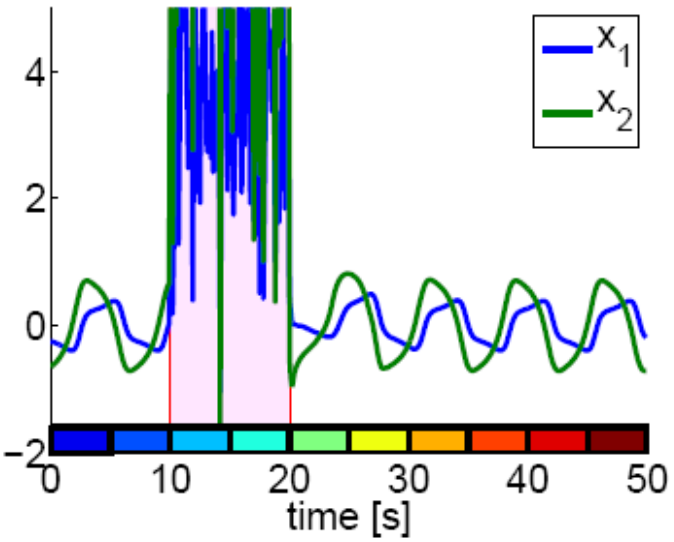
An external input „epsilon“ can switch the generated patterns in a smooth manner:







The system can autonomously recover from noise added to all spring lenghts

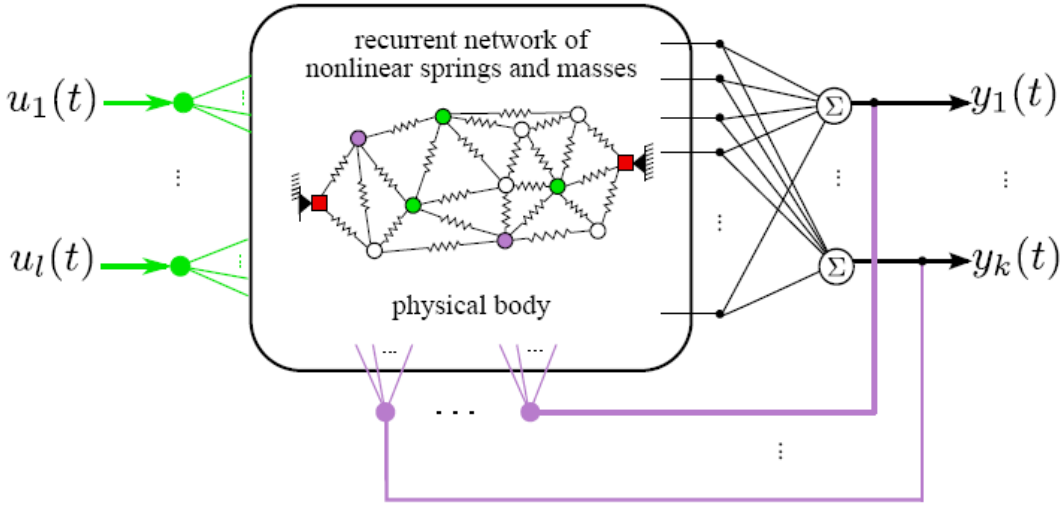


The measured lengths of the springs were superimposed by strong white noise. When it stopped, the „morphological computer“ returned immediately to the learnt generation of a limit cycle.

Elimination of teacher and teacher forcing is in principle possible

Robert Legenstein had yesterday shown in his presentation, that the teacher can often be replaced (for artificial neural networks, instead of mass-spring systems) by autonomous reward-modulated learning of readout weights

This method required only noise on readouts, plus global reward signal if system performance had recently improved.



Summary

- There exists a solid mathematical foundation for morphological computing in a class of models (mass-spring systems) that satisfy the axioms from ICMC1

--should have inputs/outputs

--should be programmable

--should have purpose/goal.

- Unusual in the underlying theory (for the case without feedback) is, that it requires a complex body structure with many different time constants.

- The increase in computational power (from Volterra series to universal analog computation) through feedback, or by training units within the body, is surprising

- Apparently this is the first time that one has found a theoretical framework for universal analog computation

Challenges

- Replace teacher for morphological computation by autonomous learning from realistic signals about changes in system performance (see yesterday‘s talk by Robert Legenstein)

- Optimize body parts so that the resulting „liquid“ supports a specific class of possible readout tasks (e.g.; periodic patterns for locomation)