Graz University of Technology

DISSERTATION

to obtain the title of

Doctor of Technical Sciences

of Graz University of Technology

Defended by

Helmut HAUSER

New Biologically Inspired Control Paradigms for Robots: Kinematic Synergies and Morphological Computation

Thesis Advisor:	Univ.Prof. DI Dr. Wolfgang MAASS
Second Reviewer:	Univ.Prof. Dr. Auke J. IJSPEERT

defended on September 7, 2010

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst, andere als die angegebenen Quellen / Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, August 22, 2010 ....
(signature)

Abstract

Biological systems exhibit an impressive ability to interact with their environment. They display high versatility in their movements, an ability to learn fast, and a remarkable robustness to noise and external perturbations. On the other hand even modern robots often look clumsy, have difficulty to learn successfully in noisy, dynamic environments, and are prone to perturbations. Therefore, robot designers seek for inspiration from nature by identifying successful strategies of biological systems and applying them to robots.

With this dissertation I want to contribute to further close the performance gap between artificial robots and their biological role models. My approach is to employ experimental data, that have been collected over a wide range of biological systems and over a variety of different tasks. I identify the underlying general strategies, formalize them by the use of rigorous mathematics, to finally apply them to robots.

My dissertation is divided into two main parts, dealing with two different biologically inspired approaches. The first one, *kinematic synergies*, describes the phenomenon in biological systems that a number of degrees of freedom (e.g., muscles, joints) are coordinated in a fixed way in order to fulfill a single task. This allows the controller, which operates the synergy, to work in lower dimensional space. Hence, the remaining control and/or learning task is much simpler. I demonstrate how kinematic synergies can be formulated mathematically and how they can be applied to balance control of a humanoid robot. Remarkably, such synergies, in conjunction with simple linear controllers, enable a humanoid robot to balance online against all kinds of unknown, dynamic perturbations with very little computational costs.

The second biologically inspired strategy, which I describe, is known as *morphological computation*. It embraces the observation that the physical body (i.e., the morphology) is not simply a device to carry the brain around, but rather that it is highly involved in computational tasks. There exit already a number of robots, which successfully implement this concept. Nevertheless, a theoretical basis for understanding the capabilities and limitations of morphological computation has been missing so far. I present different theoretical models for morphological computation, where a precise mathematical characterization of the potential computational contribution of a complex physical body is feasible. Based on these models I propose morphological computation setups, which consist of the physical body itself with static readouts and static feedbacks. These simple structures are able to emulate a surprisingly rich class of nonlinear computations (even ones with persistent memory), which map continuous input streams to continuous output streams. Remarkably, in the case of a complex, compliant body it is sufficient to add linear outputs and linear feedbacks in order to emulate nonlinear, dynamic computations. This points to an interesting property of morphological computation: it facilitates learning. By outsourcing parts of the computation to the physical body, the complex problem of learning to control a robot may be reduced to the much simpler task of finding linear output weights. This suggests that complexity and nonlinearity, typically unwanted properties of robots, are desired features in order to provide a computationally powerful physical body.

Keywords: biologically inspired, robotics, nonlinear control, nonlinear dynamic systems, morphological computation, kinematic synergies, balance control, locomotion

Zusammenfassung

Biologische Systeme sind beeindruckend in ihrer Art und Weise wie sie mit ihrer Umwelt interagieren. Sie zeigen dabei eine großes Repertoire an Bewegungen, die Fähigkeit sich schnell an neue Situationen anzupassen und eine beeindruckende Robustheit gegenüber Rauschen und externen Störungen. Auf der anderen Seite sind selbst moderne Roboter in ihren Bewegungen immer noch eher inflexibel, haben große Schwierigkeiten in einer dynamischen Umgebung zu interagieren und sie sind sehr anfällig gegenüber Störungen von außen. Daher suchen Wissenschaftler immer wieder Inspiration in der Natur. Dabei versuchen sie erfolgreiche und allgemein gültige Strategien zu identifizieren, um diese dann auf Roboter anzuwenden.

Mit dieser Dissertation versuche einen Beitrag zu leisten, die Unterschiede zwischen künstlichen Robotern und ihren biologischen Vorbilder zu verringern. Mein Ansatz ist es dabei auf Ergebnisse aus Experimenten mit biologischen Systemen aufzubauen. Ich arbeite die dahinter liegenden, allgemeinen Strategien heraus, beschreibe sie mit Hilfe der Mathematik, um sie schlussendlich bei Robotern anzuwenden.

Meine Dissertation ist in zwei Teile unterteilt. Der erste Teil beschäftigt sich mit so genannten *Kinematischen Synergien*. Der Begriff beschreibt ein Phänomen, das man in vielen verschiedenen biologischen Systemen beobachten kann. Dabei werden mehrere Freiheitsgrade (z.B.: Muskeln, Gelenke) zusammen koordiniert, um ein gemeinsames Ziel zu erreichen. Durch die daraus resultierende geringere Anzahl der Dimensionen in denen ich arbeiten muss, vereinfacht sie die eigentliche Aufgabe. Ich zeige wie man Kinematische Synergien mathematisch formulieren kann, und wie man sie zur Gleichgewichtskontrolle für humanoide Roboter verwenden kann. Die so gewonnenen Synergien, erweitert durch einfache, lineare Regler, ermöglichen es einen humanoiden Roboter online mit geringen Rechenaufwand Störungen dynamischer, unbekannter Art und Größe auszubalancieren.

Die zweite biologisch inspirierte Strategie, dich ich hier aufgreife, wird unter dem Namen *Morphological Computation* zusammengefasst. Ihr liegt die Beobachtung zugrunde, dass der physikalische Körper nicht nur dazu da ist, vom Gehirn gesteuert zu werden, sonder dass er selbst aktiv an der Regelung (z.B. um das Gleichgewicht zu halten) beteiligt ist. Es gibt bereits eine Reihe von Robotern, die in ihrer Konstruktion genau auf dieses Prinzip aufbauen. Trotz der erfolgreich Anwendung in der Praxis war es bisher noch nicht möglich eine zugehörige Theorie zu finden, mit deren Hilfe man die Möglichkeiten aber auch die Grenzen dieses Ansatzes erfassen konnte. Ich präsentiere hier daher verschiedene theoretische Modelle, mit denen man eine exakte mathematische Charakterisierung des möglichen Beitrags des physikalischen Körpers zu Berechnungen aufzeigen kann. Basierend auf diesen Modellen schlage ich Strukturen vor, die es einem ermöglichen mit Hilfe von physikalischen Körpern (Körperteilen)

Berechnungen durchzuführen. Diese bestehen dabei nur aus dem Körper an sich, und statischen Readouts und Feedbacks. Diese sehr einfachen Strukturen sind in der Lage eine außergewöhnlich große Klasse an nichtlinearen Berechnungen zu emulieren. Ein besonders interessanter Fall ist gegeben, wenn der involvierte physikalische Körper eine genügend komplexe Dynamik besitzt. Dann reicht es aus lineare Readouts und linearen Feedback zu verwenden. Als Resultat kann dann, zum Beispiel, die komplexe Aufgabe des dynamischen Interagierens eines Roboters auf die viel einfachere Aufgabe, nämlich dem Finden von linearen Readout Gewichten, reduziert werden. Eine interessante Konsequenz ist, dass daher die Körperteile von Roboter nicht statisch und starr sein sollten, wie bisher bei klassischen Robotern, sonder dynamisch und komplex.

Schlüsselwörter: Biologisch Inspiriert, Roboter, nichtlineare Regelung, nichtlineare dynamische Systeme, Morphological Computation, Kinematische Synergien, Gleichgewichtskontrolle

Acknowledgements

IT is a great pleasure to be able to thank so many people for supporting me during my dissertation. First of all, I would like to thank my supervisor Wolfgang Maass for giving me the great opportunity to work on this exciting topics and for his guidance and support over the years. I am also very grateful to Auke J. Ijspeert. In particular, I would like to thank him for his kind encouragement through out the years and the inspiring discussions, and for reviewing my dissertation and attending my defensa despite his tight schedule.

I also would like to express my deep gratitude to my colleague and co-author Gerhard Neumann, who was always a wonderful office company and a great scientist to work with. Furthermore, I want to thank my other co-authors Rolf Pfeifer and Ruedi Füchslin. It was a great pleasure to work with them and I enjoyed the numerous inspiring discussions.

It was a wonderful time to work at IGI, mostly because of my colleagues. Therefore, I would like to thank all of them for their professional support as well as for their friendships. I would like to mention Ingrid Preininger, Daniela Potzinger and Angelika Zehentner for their administrative support and their motivational speeches, as well Olivier Friedl for his enthusiastic hardware and software support, and Stefan Häusler and Lars Büsing for exciting discussions.

I take the opportunity to express my thankfulness to the people (especially to Ludovic Righetti) at the Biorobotics Laboratory at EPFL for their kind hospitality and their support during my stays.

Overall, I am deeply thankful to my parents, who have always supported me in so many ways. I also want to express my deep love and gratitude to my wonderful wife Miriam. This dissertation is also yours. I also want to thank Carmen for her valuable support, especially at the end of the dissertation.

Finally, I have to thank my three little boys Darío, Alejo and Andrés for teaching me patience, loving kindness and the very convenient ability of multitasking.

May all be happy!

Contents

Detective Del Spooner: *Human beings have dreams. Even dogs have dreams, but not you, you are just a machine. An imitation of life. Can a robot write a symphony? Can a robot turn a... canvas into a beautiful masterpiece?*

Sonny: *Can ... you?*

from 'I, robot'

CHAPTER 1

Introduction

The dream of creating artificial life has always existed. Even before elaborated artifacts were built the idea already dwelled in the mind of people. For example, in the famous Greek Iliads Hephaestus (the god of craftsmen) made talking mechanical handmaids out of gold. Throughout history people tried to build devices, which mimicked biological systems. The motivations to construct such artifacts were numerous, from the simple narcissistic joy to play god to profound scientific curiosity. However, in the 19th and 20th century people suddenly focused on building effective working machines resulting in the success story of the assembly line. The used robots were highly optimized tools, designed for very specific tasks. As a consequence, they did not look like any biological systems anymore. Although this approach was successful, the used robots were bound to a specific task and a specific environment. They were far from being called intelligent or from being comparable to biological systems.

In the last decades there has been again a growing interest in robots, which are not simple doing the same movement at the same place over and over again. This new demand for versatile robots is mostly based on new fields of applications, which have arisen. For example, todays robots should help in the household, they should support older or impaired people, or they are meant to walk autonomously in rough, unknown terrain. In general robots are supposed to interact in highly complex, dynamic and sometimes even delicate environments. Therefore, the new demands for this new types of robots are very different to the ones in an assembly line. They should exhibit versatility in their movements, adaptivity to new situation, ability to interact in noisy and dynamic environments and, last but not least, implement all of that with high energy efficiency. Biological systems face the same challenges and they master them with an astonishing ease. Therefore, scientists take again a closer look at the solutions nature provides. They try to identify successful, general strategies and adapt them for artificial robots.

One way to extract such general strategies is to take experimental data of biological systems into account. In addition, one can look at the main differences between a classical robot design and the corresponding biological model to highlight properties, which might be responsible for the difference in the performances.

For example, biological systems exhibit a much higher number of degrees than classical robots. One just has to compare the complex muscle skeleton system to the rather simple compound of rigid parts and torque driven joints. The advantage of such a high dimensional working space is the variety of solutions it offers. This implies also that the system is potentially adaptive and it can find new solutions, when confronted with a new situation. However, from the engineer's point of view such a high dimensional space represents a disadvantage. Any optimization scheme or learning algorithm struggles with a high dimensional space, since the search and learning time increases exponentially with the number of dimensions (often referred to as "curse of dimensionality"). This raises the question, why does nature does not have this problem? Looking at a number of physiological experiments over a wide range of tasks and species, for example [38, 11, 10, 13, 54, 53, 1, 35, 47], reveals an interesting strategy, the so-called *kinematic synergies*[1]. They present a fixed coordination of a fixed number of degrees of freedom. For example, in humans mixed sets of muscles and joints are working together in a coordinated way to balance. In complex biological bodies there exist a number of such synergies, which can then be combined, typically, in a linear superposition.

In Chapter 2 I seek to adapt this strategy in order to apply it to robots. I introduce a general mathematical framework to describe such kinematic synergies and I apply it to the special case of balance control for the humanoid robot HOAP-2. The results are remarkable. Our proposed kinematic synergies, only augmented by simple linear controllers, are able to maintain balance of the humanoid robot despite different external, unknown dynamic perturbations with very little online computational cost.

Another interesting property of biological systems, which is missing in classical robots, is the compliance of their body parts. Again, for a classical robot designer compliance is undesired. From their point of view they introduce unwanted complex dynamics, which makes it unnecessary more difficult to control the robot. Therefore, classical robots are built out of rigid body parts and high torque servos. Despite the disregard of the engineers, nature still uses compliant bodies. What is the advantage? One explanation is that the physical body (i.e., the morphology) of a biological system is not only a simple container for the brain, but rather that it is highly involved in computational tasks. This aspect is usually summarized under the terms *morphological computation* [42] or embodiment [44]. There exist a number of cases in biology, which support this view [43]. There have been built even robots of all kinds, which took the concept of morphological computation into their design process, e.g., [36, 61, 52, 20, 63, 62, 49]. However, so far there has been no clear mathematical framework, which was able to grasp the concept in mathematical terms. Therefore, I present in Chapter 3 and Chapter 4 theoretical models for morphological computation, which are based on rigorous mathematics. They enable us for the first time to identify the capabilities and the limitations of the computational power of physical bodies. It is even possible to determine the contribution of the morphology to the computational process. From the presented theories I derive morphological computation devices, which consist of certain types of physical bodies and some static readouts (and eventually static feed-

[1]Also sometimes called muscle synergies or movement primitives.

backs)[2]. These simple structures are able to emulate a rich class of computations, which can even include the use of persistent memory. Actually, the proposed morphological computation devices are analog devices, which are able to emulate complex, nonlinear operators, which map input streams to output streams. Hence, they are able to implement computations, which are crucial for any successful interaction in a dynamic environment.

A remarkable conclusion from the proposed theories is that, if the physical body is sufficiently complex, it is sufficient to add simple linear readouts and linear feedbacks to the physical body in order to have the full computational power. This suggests that complexity and nonlinearity, typically unwanted properties of robots, are desired features in order to provide computational powerful physical bodies.

Chapter 2 deals with the concept of kinematic synergies. It is based on the publications *"Biologically Inspired Kinematic Synergies Provide a New Paradigm for Balance Control of Humanoid Robots"* [16] and *"Nonlinear Kinematic Synergies Enable Linear Balance Control of a Humanoid Robot"* [17]. Both papers were jointly written by Gerhard Neumann (GN), Auke J. Ijspeert (AI), Wolfgang Maass (WM) and myself (HH). The optimization process with the Inverse Kinematics was implemented by Gerhard Neumann (GN), the simulations and the experiments with the real HOAP-2 robot were designed and implemented by HH (but the JPI-approach of Section 2.4.5, which was implemented by GN).

Chapters 3 and 4 is about the concept of morphological computation. The first part is based on a draft of a paper with the tentative title *"A Theoretical Foundation for Morphological Computation"* and the second part, Chapter 4, is based on a draft of a paper with the tentative title *"Morphological Computation with Explicit Feedback"*. The authors for both papers are myself (HH), Rolf Pfeifer (RP), Rudolf M. Füchslin (RF), Auke J. Ijspeert (AI) and Wolfgang Maass (WM). All of them contributed by giving feedback and by proposing experiments. The basic impulse came from WM and AI. The writing, the proofs, the implementation of the simulations as well the design of the experiments were done my myself (HH).

[2]This depends on which model we apply.

Chapter 2

Biologically Inspired Kinematic Synergies

Contents

Despite many efforts, balance control of humanoid robots in the presence of unforeseen external or internal forces has remained an unsolved problem. The difficulty of this problem is a consequence of the high dimensionality of the action space of a humanoid robot, due to its large number of degrees of freedom (joints), and of nonlinearities in its kinematic chains. Biped biological organisms face similar difficulties, but have nevertheless solved this problem. Experimental data show that many biological organisms reduce the high dimensionality of their action space by generating movements through linear superposition of a rather small number of stereotypical combinations of simultaneous movements of many joints, to which we refer as kinematic synergies in this paper. We show that by constructing two suitable nonlinear kinematic synergies for the lower part of the body of a humanoid robot, balance control can in fact be reduced to a linear control problem, at least in the case of relatively slow movements. We demonstrate for a variety of tasks that the humanoid robot HOAP-2 acquires through this approach the capability to balance dynamically against unforeseen disturbances that may arise from external forces or from manipulating unknown loads.

2.1 Introduction

Humanoid robots are constructed to have the form of a human body in order to be able to work in environments optimized for human needs. Additionally, in the near future they are meant to work with people, and human like shape would increase the possibility of acceptance of robots in human society. Nevertheless, the humanoid form carries the burden of some disadvantages compared to wheeled robots for instance. One of the biggest problem is the issue of balance, especially the case of counterbalancing unknown perturbations, which is a standard situation in a real environment and has to be solved as a prerequisite to any interaction. Due to their human structure, humanoid robots are bipedal, and have therefore a smaller support polygon (which is defined as the convex hull of the foot support area) compared to, for example, quadrupeds. In addition, two-thirds of their body mass is typically located in that part of the robot that lies two-thirds of body height above the ground [59]. Both facts contribute to the instability of humanoid robots. Furthermore, a failure of their balance control is not only bad for the robot, since a fall is likely to produce damages, but may also hurt people that interact with the robot. Therefore, a crucial point for allowing human robots to work in human environments is to find robust and effective methods for their balance control. Additionally, these solutions should induce naturally looking movements in order to increase the possibility of acceptance of humanoid robots as partners of humans.

The balance control problem of humanoid robots is known to be hard to solve due to the high dimensionality of their action space (since many degrees of freedoms, i.e., joints, are involved) and the nonlinearities inherent to any kinematic chain. Because of the importance of finding solutions to this problem, quite a bit of effort has already been invested and many approaches from different research areas have been proposed.

A first step was made by introducing the *Zero Moment Point* (ZMP) criterion [56]. It simplifies the high dimensional problem by reducing all acting forces above the foot (in case of single support, i.e., contact with the ground with only on foot) to one single force [56]. Due to physical interaction between foot and ground we get as a result of Newton's 3rd law (i.e., action-reaction), at the point where this resulting force acts, a so-called ground reaction force with opposite sign. The two dimensional point (called ZMP) on the ground, where this resulting force acts, can then be used to characterize the dynamic state of the robot: If the ZMP lies within the support polygon of the robot, the state of the robot is called dynamically stable. This so called "ZMP stability criterion" reduces the problem of stability to coordinate the limbs of the robot (i.e., apply appropriate torques through their servos) in such a way, that the ZMP stays within the support polygon[1].

While the ZMP can be calculated analytically, the position of this point can also

[1]The robot could also change the size of the support polygon by, for example, hold on to something. For a discussion of different control strategies in this context we refer to [15].

be measured by pressure sensors (actually measuring the ground reaction force). From this point of view the resulting point is called accordingly *Center of Pressure* (CoP). As Goswami demonstrated [14] the ZMP equals the CoP, since they describe the same phenomenon from different points of view. In this paper we are going to use the name CoP, since we use the pressure sensor information in combination with the support polygon to estimate the state of stability. Since the original ZMP definition has some limitations [14], other ground reference points have been proposed, for example, the *Foot Rotation Indicator* (FRI) introduced by Goswami [14] or the *Centroidal Moment Pivot* (CMP), just to name two. For a detailed discussion we refer to [45].

Other approaches have been proposed that are also based on a reduced model of the robot. For example, the *Inverted Pendulum Model,* introduced by Kajita et. al. [24], has proved to be very useful. It describes the whole robot, under some assumptions, by a linear inverted pendulum and thereby, reduces the number of dimensions. Extensions of this model have also been studied, for example, the *Three-Dimensional Inverted Pendulum Model 3D-LIPM* [25] and the *Reaction Mass Pendulum* (RMP) [29]. Although all these reduced models are useful, still, at the point of implementation one has to find control schemes which map the strategy back into the full dynamic model (as Lee and Goswami pointed out [29]). Hence, they necessarily fail to deal with unknown external perturbations, since these perturbations present a change in the dynamics of the robot.

An alternative approach to balance control is to rely on the static model, i.e., to use the kinematic model and the mass distribution of the robot. By employing a local Jacobian Pseudo-Inverse (JPI) approach on local information, like Resolved Motion Rate Control (RMRC) [58], the optimal change of the joint angles can be calculated. Some of these frameworks even allow to set priorities amongst conflicting tasks [4, 5]. Accordingly, balancing could be one of these tasks, typically with a high priority. In order to deal with unforeseen perturbations, the setup has to be used inside a feedback control loop, for example as proposed in [34]. However, a drawback of such an approach is that it calculates online inverse kinematics, which involves computationally expensive matrix inversions. In addition, such computations lack biological plausibility.

Other approaches try to solve directly the dynamic equations within constraints, which reflect the border of stability. For example, Kagami et. al. [23] proposed an online balancing scheme by solving a quadratic programming problem. However, the precise dynamic model of the robot is needed in order to apply this approach. Therefore, it could not be used in situations where the dynamic model of the robot changes due to external unknown forces, for example, introduced by picking up unknown loads or contact with the environment, which are standard situations for humanoid robots working in a human environment.

Biological organisms face similar problems, but, as experimental data suggest, employ a radically different strategy for controlling their movement apparatus with many degrees of freedom (DoF), in particular for balance control. Numerous studies from the Lab of Bizzi at MIT ([38, 11, 10]) have shown that the central nervous systems of a variety of organisms employ a modular architecture for motor control, whereby many different movements (arm movements, walking, jumping, swimming) can be constructed as largely linear (but non-negative) combinations of a rather small repertoire of movement primitives (also referred to as muscle synergies, or kinematic synergies; we use the latter term in this article).

Also recent work on whole-body movements of humans ([13, 54, 53]) show that balance control and other human body movements during standing can be understood as combinations of a small set of stereotypical kinematic synergies (each of them affects several joints). Experiments, where humans where asked to bend their upper trunk, while recording the angles of the ankle, hip and knee, revealed after a Principal Component Analysis (PCA) of these angles, that already the first principle component can explain over 99% of the total angular variance [1]. This suggests that a set of muscles (multiple degrees of freedom) are controlled by a low dimensional (possibly even one dimensional) variable. Other experiments suggest that this principle of kinematic synergies is present over a wide range of different movements like reaching and grasping [35], upper-arm movement [47] and making a step [57]. Hence, kinematic synergies seem to present a general strategy biological organisms apply.

We are especially interested in humanoid balance control, Therefore, we apply this basic modular strategy (based on kinematic synergies) to balance control and demonstrate how it can be used in the case of the humanoid robot HOAP-2. The kinematic synergies were calculated offline by an optimization process based only on the *static* model (kinematics and masses) of the robot[2]. However, we are able to demonstrate that the concept of kinematic synergies, when plugged into a linear control loop, can provide a powerful scheme for *dynamic* balance control.

In the next section we define kinematic synergies. Section 2.3 explains how to construct and use kinematic synergies for balance control of the humanoid robot HOAP-2. In Section 2.4 we present a number of experiments with the simulated and the real HOAP-2.

2.2 Formal Definitions of Kinematic Synergies

In this section we define kinematic synergies in order to reduce high dimensionality and nonlinearities. Typically, humanoid robots have a high number of degrees of freedom (DoF), namely joints. We interpret kinematic synergies (KS) as a way to reduce the DoF by putting a defined set of joints under the regime of one controlling parameter, which we refer to as the KS-parameter s. We define a kinematic synergy as a nonlinear

[2]This optimization process is closely related to the Jacobian Pseudo-Inverse approaches [48], however, the computations are only needed for the (offline) construction of the synergies and not during online control.

mapping $\mathbf{\Phi}$ of the KS-parameter $s \in \mathbb{R}$ to a fixed number of m degrees of freedom (joints).

Definition 2.1. A *kinematic synergy* (KS) is a function $\mathbf{\Phi} := \mathbf{\Phi}(s)$ which maps the KS-parameter $s \in \mathbb{R}$ onto a m dimensional vector of joint angles $\mathbf{q}^{KS} = \mathbf{\Phi}(s)$:

$$\mathbf{\Phi} : \mathbb{R} \rightarrow \mathbb{R}^m . \tag{2.1}$$

The superscript ^{KS} denotes the subset of m joints, which are controlled by the *KS*. The total number of joints of the robot is denoted by n. Further, we define the function φ

$$\varphi : \mathbb{R}^m \rightarrow \mathbb{R}^n \tag{2.2}$$

to embed the m-dimensional subspace spanned by $\mathbf{\Phi}$ into the n-dimensional space of all joints of the robot. This embedding copies the angles of all joints affected by $\mathbf{\Phi}$ and leaves the remaining joints constant.

A *KS* is typically applied in order to control a low-dimensional, or even one-dimensional, variable $y \in \mathbb{R}^l$. In general the output y depends on all n joint positions $\mathbf{q} \in \mathbb{R}^n$ of the robot and can be described by a nonlinear function $\mathbf{f}(\mathbf{q})$

$$\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^l. \tag{2.3}$$

We want the *KS* to control the output $y = (\mathbf{f} \circ \varphi \circ \mathbf{\Phi})(s)$. In the case of balance control, the function $\mathbf{f}$ represents the nonlinear relationship between all joints of the robot and a ground reference point like the CoP. We will use two *KS* $\mathbf{\Phi}_x$ and $\mathbf{\Phi}_z$ for the two dimensions of the CoP. Therefore, in this particular case each *KS* is used to control a one-dimensional output ($l = 1$).

Since such a *KS* affects m degrees of freedom that depend just on a one dimensional parameter s, we can impose further constraints on the function $\mathbf{\Phi}$. A reasonable choice for such a constraint is a linear relationship between the controlling parameter s and its corresponding output y. This reduces nonlinearities, inherent to kinematic chains, and hereby facilitates controlling and learning. Hence, we are particularly interested in the following type of *KS*:

Definition 2.2. A *linearizing kinematic synergy* is a kinematic synergy according to Definition 2.1, which has a linear relationship between its controlling parameter s and the corresponding (to be controlled) output y

$$y = (\mathbf{f} \circ \varphi \circ \mathbf{\Phi})(s) = k \cdot s, \qquad k \in \mathbb{R} . \tag{2.4}$$

We restrict our attention in this article to such *linearizing KS*, to which we simply refer as *KS*. Figure 2.1 depicts Equation 2.4.

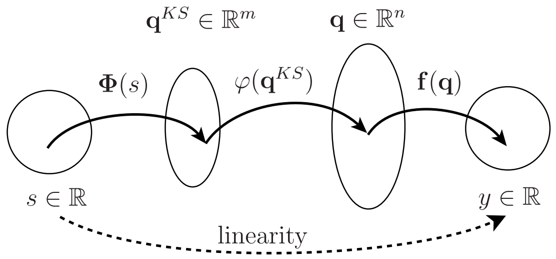

$\mathbf{q}^{KS} \in \mathbb{R}^m$
$\mathbf{q} \in \mathbb{R}^n$
$\mathbf{\Phi}(s)$
$\varphi(\mathbf{q}^{KS})$
$\mathbf{f}(\mathbf{q})$
$s \in \mathbb{R}$
$y \in \mathbb{R}$
linearity

Figure 2.1: Scheme for the composition of the functions φ and $\mathbf{f}$ according to (2.2) and (2.3) with the linearizing kinematic synergy $\mathbf{\Phi}$, which fulfills 2.4.

For a better understanding we provide some additional remarks:

1. As stated above the property of linearity in Definition 2.2 reduces inherent nonlinearities. But Equation 2.4 presents a static mapping, and therefore it will only linearizes the static part (linearization at $\dot{\mathbf{q}} = \mathbf{0}$, $\ddot{\mathbf{q}} = \mathbf{0}$) of the whole dynamic model of the robot. Nevertheless, it will reduce nonlinearities in the dynamic regime to some extent too, since the dynamic part is coupled with the static part of the differential equations.

2. The controlled variable y is one-dimensional, but is controlled by $m > 1$ joints. Hence, we have additional redundant degrees of freedom and therefore, we are free to impose additional constraints on the *KS*. Naturally, the choice will depend on the task for which the *KS* are constructed. In our case of balance control we used constraints to assure double support and an upright posture (used in the optimization process described in Section 2.3.1).

3. *KSs* are calculated offline for each robot (see Section 2.3.1) and subsequently fixed during simulation as well as when used with the real robot. In a biological interpretation we assume the *KSs* to be found by evolution.

4. The presented framework was kept as simple as possible. Various extensions, which lead to a better performance for particular tasks, are possible. For example, one could define a two dimensional kinematic synergy (i.e., $s \in \mathbb{R}^2$ and $y \in \mathbb{R}^2$) or time-varying *KSs* ($\mathbf{q}^{KS} = \mathbf{\Phi}(s, t)$), which depend on a cyclic movement, for example, to be used in a walking cycle.

2.3 Using Kinematic Synergies for Balance Control of the Humanoid Robot HOAP-2

In this section we show in detail how to use kinematic synergies for balance control of the humanoid robot HOAP-2, see Figure 2.2A. The robot has $n = 25$ degrees of freedom (rotational joints). Its structure can be seen in Figure 2.2B. The goal is to construct *KSs* for balance control in double support. Therefore, we have to decide **(a)** what output function $\mathbf{f}$ and output variables y we are going to use, **(b)** which subset of m joints we put under the regime of the *KSs* and **(c)** what additional constraints we are going to apply to construct the *KS*s:

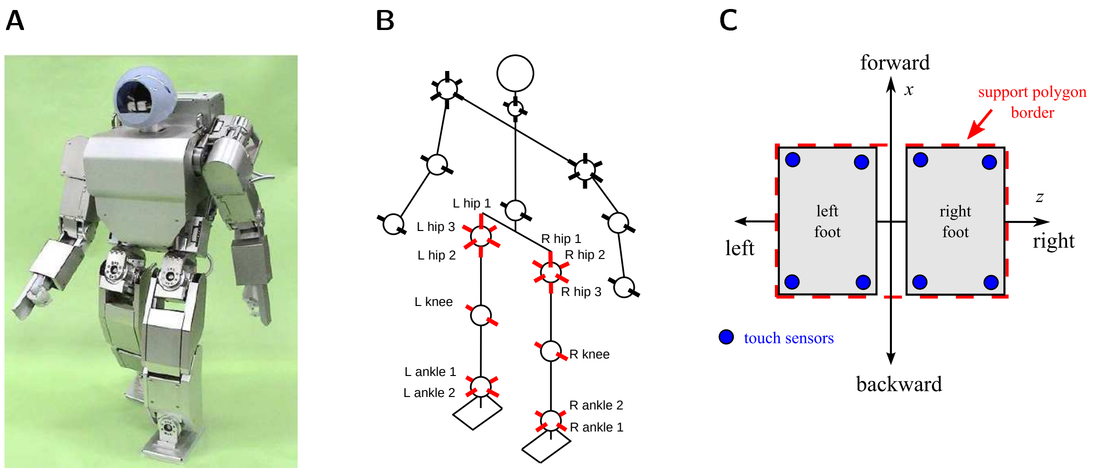


Figure 2.2: **(A)** The real HOAP-2 robot and **(B)** its schematic structure. The red marked and labeled joint rotation axes are controlled by the kinematic synergies $\mathbf{\Phi}_x$ and $\mathbf{\Phi}_z$. **(C)** Support polygon on the support surface for the robot, including the touch sensors, which are used to measure the center of pressure ($mCoP$). Black arrows indicate the x dimension (forward/backward: range 9.5 *cm*) and z dimension (left/right: range: 14.3 *cm*) for movements of the center of pressure.

(a) For balance control a natural choice for the function $\mathbf{f}$ is a ground reference point. These points are mathematically defined and can be analytically derived, but in practice, they are estimated via pressure sensors. Therefore, we will denote the reference point measured by the pressure sensors as *measured Center of Pressure* ($mCoP$). HOAP-2 has four of such sensors per feet, located at the corners (see Figure 2.2C). Since a *KS* is defined as a static mapping, we use the static version of the $mCoP$ to construct our *KS*. In the static case (zero joint velocity $\dot{\mathbf{q}}$ and zero joint acceleration $\ddot{\mathbf{q}}$) the $mCoP$ coincides with the *projected Center of Mass* (pCoM). Therefore, we chose the pCoM as output function $\mathbf{f}$. Since the pCoM is a two dimensional point on the supporting surface, we split it up into its two dimensions $y_x = \mathrm{pCoM}_x$ and $y_z = \mathrm{pCoM}_z$ and define two separate *KS*s, namely $\mathbf{\Phi}_x$ and $\mathbf{\Phi}_z$, in order to control these one-dimensional outputs y_x and y_z.

(b) Next, we have to decide what joints are placed under the regime of our *KSs*. A natural choice for balance control is to use all $m = 12$ leg joints (three hip joints, one knee joint and two ankle joints for both legs). Their corresponding rotational axes are highlighted in red in Figure 2.2B. The additional surplus of joints are free to be used for other tasks (grasping, lifting weights, tracking objects, etc.). Their movements clearly will change the pCoM too, but as we show later in Section 2.4, our approach is able to deal with that in a natural way.

(c) Finally, we choose some additional constraints (next to the linearity property) for the *KS*s, which are used for the optimization process described in the next subsection. Suitable constraints for balance control are to keep the upper body as upright as possible and to maintain double support.

2.3.1 Calculating Kinematic Synergies with Inverse Kinematics

In this section we describe the process to obtain the desired *KSs* in detail. All calculations are based only on the kinematic model of the robot including the mass information (no dynamical information like the inertia matrices is needed). The *KSs* were constructed offline and subsequently fixed during control action.

We defined an initial posture $\mathbf{q}_{init}$ (see Figure 2.4-A). This posture resulted (for the case of a horizontal support surface) in a pCoM at the center of the support polygon. We used a posture with wide-spread arms in order to avoid self collision when moving. The *KS*-parameters s_x and s_z were rescaled such that the values -1 and $+1$ corresponded to the borders of the support polygon. Therefore, the region of acting without falling was (for the case of a horizontal support surface) $s_x/s_z \in [-1, +1]$ for both dimensions x and z, see red-dashed lines in Figure 2.2C. We additionally set the origin of the coordinate system for the pCoM to the center of the support polygon and therefore, the resulting outputs in the initial posture were $\mathbf{f}_x(\mathbf{q}_{init}) = \mathbf{f}_z(\mathbf{q}_{init}) = 0$.

We will only describe the procedure for $\mathbf{\Phi}_x$. The second kinematic synergy $\mathbf{\Phi}_z$ was obtained in a similar manner. The *KS* was implemented as look-up table which maps the *KS*-parameter $s_x \in [-1, +1]$ to joint angle offsets (with regard to the initial posture)[3], i.e., $\Delta\mathbf{q}_x = \varphi(\mathbf{q}_x^{KS}) - \mathbf{q}_{init}$. Note that the look-up table represents a discretized version of a *linearizing kinematic synergy* as defined in Definition 2.2. In order to obtain joint angle offsets in between the table entries a linear interpolation was used. We used joint angle offsets instead of absolute joint angles in order to be able to use a

[3]The function φ is used to project the m-dimensional vector $\mathbf{q}_x^{KS}$ into the n-dimensional space of all joints.

linear superposition (as biological data suggest) of both KSs, i.e., $\Delta\mathbf{q} = \Delta\mathbf{q}_x + \Delta\mathbf{q}_z$. Although, the problem is due to the kinematic chains nonlinear, we will show that a linear superposition is valid for a wide range of postures. The linear superposition allows us to use two separate simple KS, which depend only on a one-dimensional KS-parameter, and which can be constructed independently[4].

In order to construct the look-up table, we divided the range of the KS-parameter s_x over the support polygon into 80 points. Therefore, the distance between two neighboring points represents 9.5 cm / 80 $\approx$ 0.12 cm in the pCoM space, which corresponds to a step of $\Delta s_x = 0.025$ in the KS-parameter space.

The construction of the KS consisted of two alternating optimization steps (see optimization scheme in Figure 2.3). Starting from $\mathbf{q}_{init}$ and $s_x = 0$, the first optimization step was used to move the pCoM of the robot to the next point y'_x of the look-up table (located 0.12 cm in x-direction from the origin). In addition, the optimization tried to keep the upper part of the body upright. An inverse kinematics algorithm based on the Jacobian Pseudo-Inverse (JPI) [48] was used to calculate the joint movement. Therefore, the applied Jacobian matrix consisted of two $3 \times m$ sub-matrices, the Jacobian for the position of the pCoM and the Jacobian for the rotation of the torso. However, due to the movement calculated by this optimization, the position of the right foot relative to the left foot tended to change. This should be avoided in order to prevent the robot from falling. Therefore, a second JPI optimization step (see Figure 2.3) was used to move the right foot back into its original position relative to the left foot. For this optimization the same Inverse Kinematics algorithm was applied using only the 6 joints of the right leg.

[4]Without this property, one would have to construct one single KS with a two-dimensional KS-parameter, i.e., $s \in \mathbb{R}^2$.

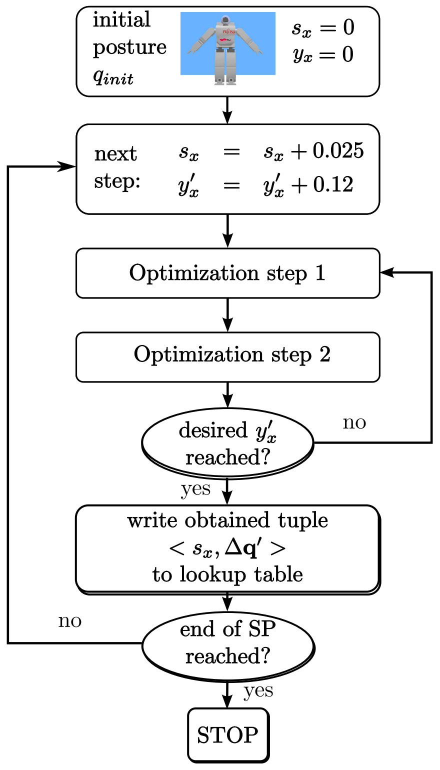


Figure 2.3: Scheme of the construction process for the look-up table for the KS $\mathbf{\Phi}_x$ in the form $< s_x, \Delta\mathbf{q} >$. Optimization step 1 moves the pCoM in the desired direction to y'_x, while keeping the trunk in an upright position. Optimization step 2 keeps the feet at the initial positions.

These two previously described steps were iterated until the desired output value y'_x was reached. Subsequently, the joint angle offsets to the initial posture were stored in the look-up table and, now starting from the new joint position, the next entry of the look-up table was calculated. The same process was applied for the opposite direction (i.e., for s_x from 0 to -1). This finally led to a look-up table for the range $s_x \in [-1, +1]$ which mapped the KS-parameter s_x to joint angle offsets.

Figure 2.4 presents four typical postures for different KS-parameter pairs $[s_x/s_z]$. The center of the figure shows the support polygon (gray area) and the coordinate system of the KS-parameters. The yellow circles (A-D) represent the postures in the KS-parameter space. The corresponding screenshots can be seen in the corners of the figure.

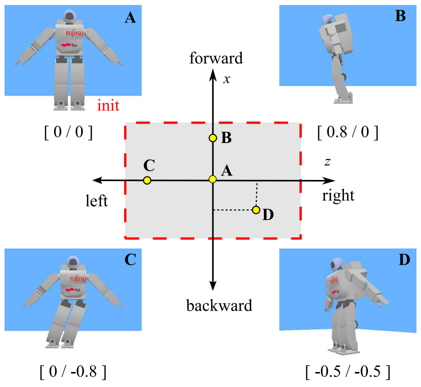


Figure 2.4: Typical postures of the simulated HOAP-2 resulting from the *KSs* $\mathbf{\Phi}_x$ and $\mathbf{\Phi}_z$ for different *KS*-parameters. The center of the figure shows the defined coordinate system for the *KS*-parameters s_x and s_z. The gray shaded area indicates the support polygon (SP) of our robot standing with both feet on the ground. The red dashed lines depict the limits of the SP and correspond to the values $s_x = \pm 1.0$ and $s_z = \pm 1.0$. The yellow points show typical postures in the *KS*-parameter space. Corresponding postures can be seen in the corners (labeled from **A** to **D**). The used *KS*-parameters $[s_x/s_z]$ can be seen below the screenshots. Screenshot **[A]** shows the initial posture $\mathbf{q}_{init}$ ($s_x = s_z = 0$ / at the origin) **[B]** shows the robot bending forward with $s_x = 0.8$ and $s_z = 0.0$, while in **[C]** the robot is bending to the left (with $s_x = 0.0$ and $s_z = -0.8$). Screenshot **[D]** presents a combination of both kinematic synergies with $s_x = -0.5$ and $s_z = 0.5$.

Figure 2.5A shows the mapping of the *KS*-parameter s_x to the outputs y_x =pCoM$_x$ and y_z =pCoM$_z$ for the *KS* $\mathbf{\Phi}_x$. We can identify a linear relationship between s_x and y_x, whereas the second output dimension y_z is unaffected by s_x. The same plot for the *KS* $\mathbf{\Phi}_z$ is shown in Figure 2.5B.

A graphical representation of the joint angle offsets over the range of the *KS*-parameter spaces (from -1 to $+1$) for the kinematic synergies $\mathbf{\Phi}_x$ and $\mathbf{\Phi}_z$ is presented in Figure 2.6. Similar to their biological prototypes (see Figure 4 in [10]), the two *KSs* largely affect disjoint sets of joints. The joints mainly responsible for the movement in x-direction are orthogonal to the joints mainly responsible for the z-direction. Note that the human muscle-skeleton system exhibits, although more complex, a similar structure. This orthogonality suggests to combine the two *KSs* linearly which is done by summing up the initial posture and the two joint angle offsets $\mathbf{q}_L = \mathbf{q}_{init} + \Delta\mathbf{q}_x + \Delta\mathbf{q}_z$.

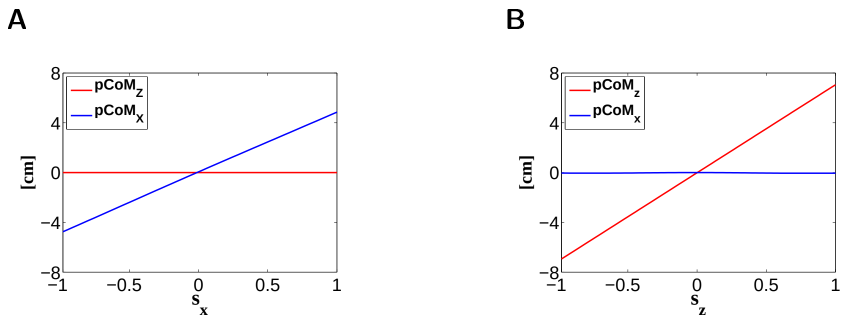


Figure 2.5: **(A)** The plot shows the mapping of the *KS*-parameter s_x to the outputs $y_x = \text{pCoM}_x$ and $y_z = \text{pCoM}_z$ for the *KS* $\mathbf{\Phi}_x$. While the relationship between s_x and y_x is linear (as demanded by the definition of a *linearizing kinematic synergy*, Equation 2.4), y_z is nearly unaffected by s_x. **(B)** The same plot for the second *KS*-parameter s_z.

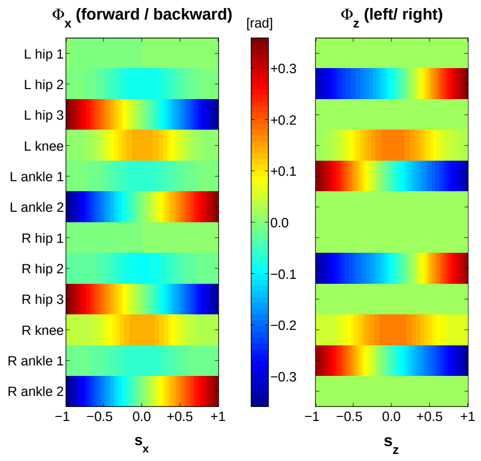


Figure 2.6: Graphical representation of the *KS*s $\mathbf{\Phi}_x$ and $\mathbf{\Phi}_z$. Shown are the joint angle offsets (in color coding) for the kinematic synergies $\mathbf{\Phi}_x$ (moves the pCoM forward/backward) and $\mathbf{\Phi}_z$ (moves the pCoM left/right) for the HOAP-2 over the range $[-1, +1]$ for the *KS*-parameters s_x and s_z. Note that these two *KSs* affect largely disjoint sets of joints.

In order to show the validity of the linear superposition of the two *KS*s, we evaluated empirically the deviation of the actual pCoM $< \mathbf{f}_x(\mathbf{q}_L), \mathbf{f}_z(\mathbf{q}_L) >$ from the case of perfect linear superposition $< \mathbf{f}_x(\mathbf{q}_{init} + \Delta\mathbf{q}_x), \mathbf{f}_z(\mathbf{q}_{init} + \Delta\mathbf{q}_z) >$. The deviations for the whole support polygon can be seen in Figure 2.7. Except for extremal cases, where the pCoM is located at a corner of the support polygon, the deviations from linearity are quite small.

Note that the described optimization procedure is closely related to standard JPI approaches. However, these approaches are typically used for online control, involving computationally expensive real-time calculations. With the use of kinematic synergies most of this computational load can be transferred to the offline optimization scheme. As a consequence, and as we will demonstrate later, without a significant loss of performance the robot can be balanced with very little computational power.

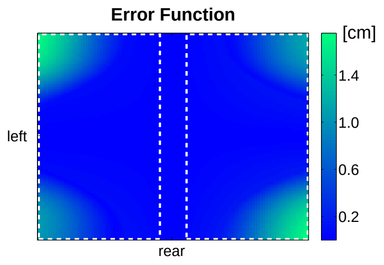


Figure 2.7: Empirical evaluation of the validity of the linear superposition of the *KSs* $\mathbf{\Phi}_x$ and $\mathbf{\Phi}_z$. We calculated the deviation of the actual pCoM $< \mathbf{f}_x(\mathbf{q_{init}} + \mathbf{\Delta q_x} + \mathbf{\Delta q_z}), \mathbf{f}_z(\mathbf{q}_{init} + \Delta\mathbf{q}_x + \Delta\mathbf{q}_z) >$ from the case of perfect linear superposition $< \mathbf{f}_x(\mathbf{q}_{init} + \Delta\mathbf{q}_x), \mathbf{f}_z(\mathbf{q}_{init} + \Delta\mathbf{q}_z) >$. The Euclidean norm of the deviations is shown in color code for the whole support polygon. Except for extremal cases, where the pCoM is located at a corner of the support polygon, the deviations from linearity are quite small. The white dotted lines depict the contours of the feet.

2.3.2 From Statics to Dynamics by Using Linear Controllers

The kinematic synergies $\mathbf{\Phi}_x$ and $\mathbf{\Phi}_z$ were constructed using the pCoM as output function, and therefore they were based on the static model of the robot. However, the robot can only estimate the $mCoP$ with its pressure sensors[5], which is also affected by the dynamics of the robot. Nevertheless, we are still able to use the obtained *KSs* in a dynamic context if following assumption holds:

***Assumption*:** The robot moves sufficiently slowly such that

$$mCoP \approx \text{pCoM}.$$

As we will demonstrate in this section, the assumption allows us to use simple linear controllers in conjunction with the *KS*s. Due to the assumption we are in principle limited to "sufficiently slow" movements. However, we will demonstrate in our experiments that a wide range of unknown external forces can be counterbalanced by our approach, despite this limitation.

We now explain how the kinematic synergy $\mathbf{\Phi}_x$ can be used in combination with a linear controller for balancing the robot in x-direction (forward/backward). For the

[5]In our simulations of the HOAP-2 we also used simulated pressure sensors to calculate the $mCoP$.

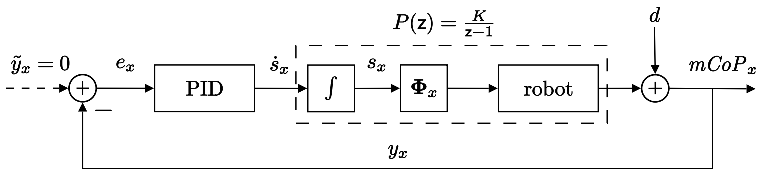


Figure 2.8: Closed control loop for the kinematic synergy $\mathbf{\Phi}_x$. Since we want to have the $mCoP_x$ at the center of the support polygon, the reference point is set to $\tilde{y}_x = 0$. The external perturbation d results from external forces and/or model uncertainties.

other KS $\mathbf{\Phi}_z$ the process is similar. As long as the assumption holds, the function from the time derivative $\dot{s}_x$ of the KS-parameter to $mCoP_x$ can be approximated by a linear transfer function

$$P(\mathsf{z}) = \frac{K}{(\mathsf{z}-1)} \ , \tag{2.5}$$

with $K \in \mathbb{R}^+$ and with z being the time shift operator for discrete systems [39]. The denominator polynomial represents an integrator (one pole at $\mathsf{z} = +1$), which integrates the velocity $\dot{s}_x$ of the KS-parameter to obtain s_x.

As long as the dynamical effects are small enough, they can be seen as uncertainties in the linear model of Equation 2.5. Already a simple linear feedback controller can handle these small uncertainties. In order to obtain a closed control loop we define a feedback error

$$e_x := \tilde{y}_x - y_x \tag{2.6}$$

with $\tilde{y}_x$ being the desired output value and $y_x = mCoP_x$. The goal is to prevent the robot from falling. Therefore, the $mCoP_x$ should stay close to the center of the support polygon. Since we have defined the center of SP at the origin, see Figure 2.2C, the desired value $\tilde{y}_x$ is set to 0.

We can now use a general standard PID controller to get the controller output $\dot{s}_x$

$$\dot{s}_x = K_P e_x + K_I \int e_x dt + K_D \frac{de_x}{dt} \quad , \tag{2.7}$$

where K_P, K_I and K_D are the positive PID controller parameters. Figure 2.8 shows the described closed control loop for the kinematic synergy $\mathbf{\Phi}_x$. Since the plant (see Equation 2.5) already contains an integrator, the use of PD controllers ($K_I = 0$) is sufficient. For the KS $\mathbf{\Phi}_z$ we used a similar control loop, which worked independently from and in parallel to the first control loop.

We have described the control scheme to control around a set point ($\tilde{y}_x = \tilde{y}_z = 0$). However, the control loop can also be used to move the $mCoP$ on any desired time varying trajectory[6], i.e., $\tilde{y}_x(t)$ and $\tilde{y}_z(t)$. This is useful in many applications. For

[6]See for example Section 2.4.2.

example, for the purpose of initiating a walking cycle, the robot has to move its $mCoP$ under the future supporting foot in order to be able to raise the other leg without falling.

The controller parameters used in the experiments were empirically found to have a reasonable performance. As we demonstrate (see Subsection 2.4.4) there is a wide range of appropriate controller parameters and therefore the choice of the parameters is not critical.

Linear and nonlinear control theory offers a number of possible improvements for the controllers, for example, adaptive control (see [2]) or robust control schemes, optimal control and different trial and error approaches to find good control parameters (see for example [28]). Even higher order controllers or different control structures than in Figure 2.8 could be used. However, in order to illustrate the capability of using kinematic synergies for balance control, we only use the previously presented, simple PID controllers.

2.3.3 Examination of Different Possible Perturbations

Lets take a closer look at possible perturbations d for the proposed control loop (Figure 2.8). We will distinguish between three different kinds of perturbations:

1. *Model perturbations:* Since we obtained our *KSs* from the static model of the robot, unmodeled dynamics, which will always be present to some extent, result in model perturbations.

2. *Internal perturbation*s: The $mCoP$ is also influenced by movements of joints, which are not under the control of the kinematic synergies. For example, if our humanoid robot uses the presented *KSs* for balancing and additionally moves a heavy weight with its arms, this movement will also change the $mCoP$ position. Note that the proposed control loop does not need any information about the movements of these joints.

3. *External perturbations:* For example pushes, pulls, contact with the environment or a moving support platform.

Since a standard feedback control loop has the property to suppress the perturbations d, our approach works for a wide range of tasks. As shown in our experiments [16], these tasks include counteracting external forces, following trajectories, compensating for forces introduced by movements of the limbs of the robot or even a mixture of these tasks. If the perturbation is too large, the assumption ($mCoP \approx$ pCoM) might be violated and the controller will therefore not be able to compensate the resulting error anymore. Yet, as our experiments show, the proposed system is capable to react appropriately to a wide range of perturbations.

2.4 Experiments

We conducted experiments with our proposed approach to demonstrate the variety of possible applications. We show that kinematic synergies with linear controllers empower a humanoid robot to counterbalance different kinds of dynamic perturbations. In our first experiments the robot had to counteract a moving support surface (platform where it stood on) and abrupt unforeseen external forces simultaneously (see Subsection 2.4.1). In the next experiment the robot had to move its $mCoP$ along a desired trajectory even when additional perturbations were introduced by manipulating an unknown weight (see Subsection 2.4.2). Subsequently, we show that the approach can be easily extended to balancing in single support (the robot only stood on one foot, see Subsection 2.4.3) and that robustness against parameter changes is an inherent property (Subsection 2.4.4). Furthermore, we compare our approach to an online Jacobian Pseudo-Inverse (JPI) algorithm (Subsection 2.4.5). Finally, we demonstrate that our approach can be easily transferred from the simulation to the real robot without any special precautions (Subsection 2.4.6).

All simulations were implemented in the robot simulation software Webots [37]. A detailed model of the dynamics of the HOAP-2 robot, based on data provided by the vendor Fujitsu, was used. The basic simulation time step was set to 2 ms and the time steps for the control loops were set to 8 ms. In the general setup we had two kinematic synergies ($\mathbf{\Phi}_x$ and $\mathbf{\Phi}_z$), which were used within two separate control loops. They reacted independently from each other on their corresponding output dimension x and z. In dependence on their errors e_x and e_z, both linear controllers calculated the velocities $\dot{s}_x$ and $\dot{s}_z$ of their KS-parameters. The velocities were integrated numerically to obtain s_x and s_z, which were then mapped via the look-up table into joint angle offsets. Subsequently, these joint angle offsets were linearly combined as described in Subsection 2.3.1 to get the actual joint target angles. Finally, these angles were transformed into torques by local PD controllers[7] at the servos. Note that there are accompanying videos on my homepage available (http://www.igi.tugraz.at/helmut/thesis).

2.4.1 Moving Support Platform (Surfboard Task)

In this task we simulated the HOAP-2 robot standing on a movable support platform (surfboard). The surfboard could rotate about the x-axis with angle Θ_x and about the z-axis with the angle Θ_z. Typical scenarios of the setup can be seen in Figure 2.9.

[7]Note that these are the hardware controller of the servos and not the controllers from our proposed control loops.

	x-direction		z-direction	
	forward	backward	left	right
without control	$+10.6°$	$-8.6°$	$-14.3°$	$+14.3°$
with control	$+20.1°$	$-22.3°$	$-26.4°$	$+26.4°$
improvement	89.6%	159.3%	84.6%	84.6%

Table 2.1: Results for very slow (quasi-static) movements of the support platform on which the robot was standing. The first row shows the tilt angles of the support surface at which the robot lost its balance when no balance control was applied. The second row shows the tilt angles when our linear controller combined with a *KS* was used. The controller enables the robot to tolerate about twice the tilt angle.

In a first experiment, we tested the capability of our linear controllers combined with the *KSs* for the quasi-static case. We tilted the surfboard on which the robot was standing and determined at which tilt angle the robot fell over. The tilting was carried out very slowly (quasi-static) and was done separately for the x- and z-direction. Table 2.1 shows that our control strategy allows the robot to tolerate about twice the tilt angle without loosing its balance – compared with a robot which does not change its posture in an adaptive manner.

Next we considered the case where the surfboard was tilted dynamically in random directions. The random trajectories for the angles Θ_x and Θ_z were generated independently from each other by smoothing (by the use of a discrete low-pass FIR-filter[8]) random trajectories of jumps (steps) with random amplitudes and random durations. Typical resulting trajectories are presented in Figures 2.10A and 2.10B.

In addition to the random movement of the surfboard, unforeseen external forces (for example these forces could arise from wind or contact with other objects) were applied to the torso of the robot at various points in time. We designed this scenario in order to show that our proposed approach is able to deal with different kinds of external perturbations simultaneously. Furthermore, control strategies that require knowledge of the dynamic model of the robot are inapplicable in this scenario, because the external forces change the dynamic model of the robot in an unknown, online manner. Figure 2.10 shows the results when an external force $W1 = [0, 0, 5]^T\ N$ (a force from the right side) was applied at the torso of the robot during the interval [5s, 10s], and another external force $W2 = [5, 0, -5]^T\ N$ (a force from the right and the back) was applied during the interval [15s, 20s] (we shaded these two time intervals in gray). Note that the onsets of the winds were abrupt (i.e., a step function in time) and therefore represented highly dynamical perturbations to the system.

Typical trajectories of the *mCoP* for the described setup, with and without balance control, are shown in Figures 2.10C and 2.10D. Without balance control, the robot

[8]The used FIR-filter had three poles at 0.997.

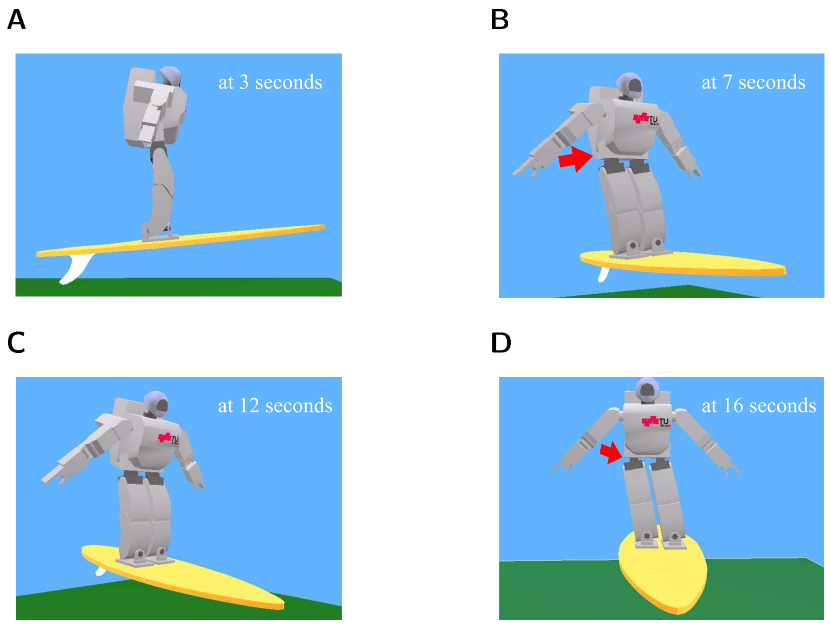


Figure 2.9: Screenshots of the posture of the (simulated) HOAP-2 at 4 time points during the balancing experiment with the random moving support surface (surfboard) and external perturbations (winds). In Figure **(B)** the wind $W1$ was blowing from the right (point of view of the robot; red arrow). As a consequence, the robot was leaning against the wind in order to move its $mCoP$ back into the middle of the support polygon. In Figure **(D)** another wind $W2$ (red arrow) was blowing from the right and the back, resulting in a diagonal force. Again, the robot responded properly to this online modification of its dynamic model.

lost balance after 16s (indicated by a black star in Figures 2.10C and 2.10E), whereas with our controllers, balance was maintained. The error signals for both dimensions x and z can be seen in Figures 2.10E and 2.10F. Note that both perturbations, the movements of the surfboard and the external forces, are *external perturbations*. In addition, as the setup was dynamic, inherent *model perturbations* were present too. With this experiment, we demonstrated that our approach is able to react online against a mixture of different types of unforeseen perturbations.

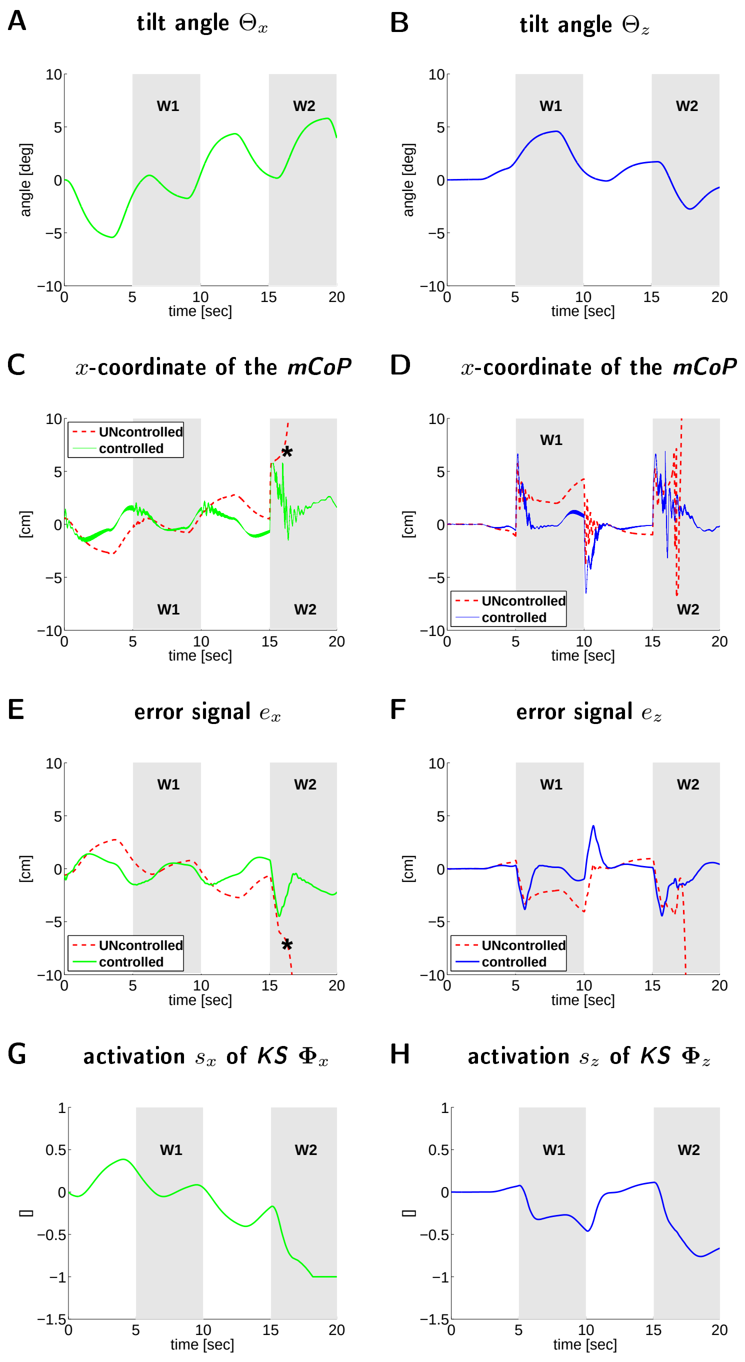


Figure 2.10: Results of the experiment with a moving support platform (surfboard) and unexpected external forces (wind) $W1$ and $W2$. The balance of the HOAP-2 is controlled by two linear controllers combined with the kinematic synergies. Without balance control (red dashed line in **C** and **D**) the $mCoP$ left the support polygon after 16s (in response to the wind $W2$), and the robot fell over. With balance control (solid lines) the stability of the robot was maintained in spite of these unexpected external forces.

2.4.2 Trajectory Following

In contrast to the preceding experiment, where the robot was controlled around a set point ($\tilde{y}_x = \tilde{y}_z = 0$), we demonstrate in this experiment that our approach can also be used to follow any desired time varying trajectory of the *mCoP*. Trajectory following is of interest for many applications, for example, to initiate a walking cycle by moving the *mCoP* under the future single supporting foot.

In this experiment the robot stood on flat ground. It was supposed to follow a desired trajectory of the shape of a figure "eight", see Figure 2.11C. Figures 2.11A and 2.11B show the same desired trajectory (gray, dash-dotted curves), split up into its dimensions x and z. In addition to following the desired trajectory, the robot manipulated a heavy weight (20% of the robot's weight) with its left arm. Note that the arm joints were not under the control of the *KSs,* but their movements perturbed the *mCoP*. This represented an *internal perturbation* as described in Subsection 2.3.3. The impact of the arm movement on the *mCoP*, when no control action was applied, is shown by the red dashed curves in Figures 2.11A and 2.11B. The same figures show that the robot was able to follow the desired trajectory despite the arm movement (the green and blue curves show the actual *mCoP*). The control loops reacted with sinusoidal time courses of the *KS*-parameters in order to follow the desired trajectory. In addition, as response to the arm movement, the control loops introduced time varying offsets in the *KS*-parameters (see Figure 2.11D). The first row (**A**) of Figure 2.12 shows the robot during the arm movement without balance control. The second row (**B**) of Figure 2.12 shows the robot with active balance control. Note that at the end of the simulation (last screenshot on the right), the robot leaned to the right in order to keep its *mCoP* at the center of the support polygon.

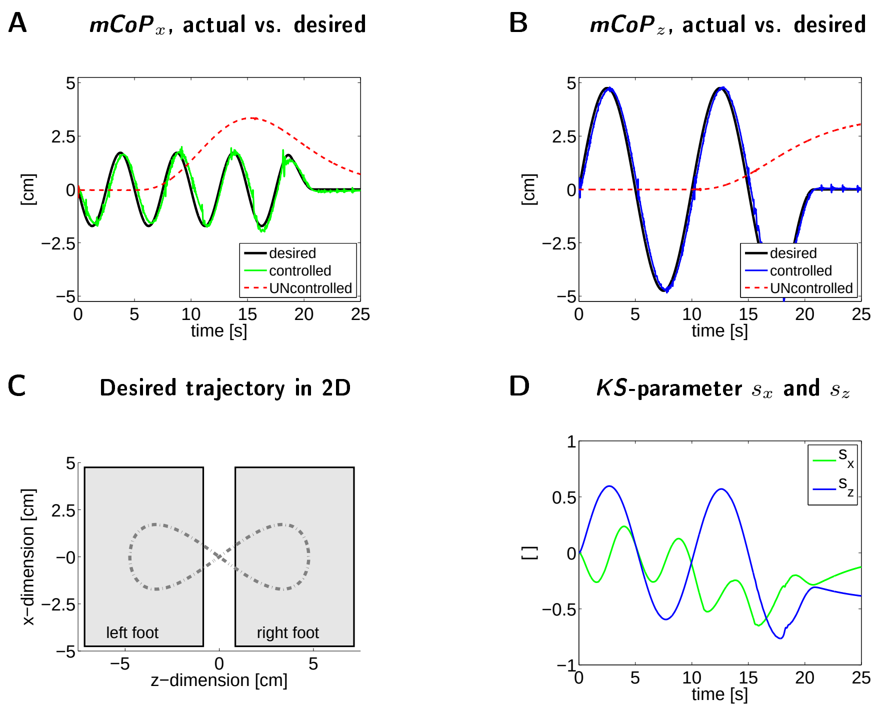


Figure 2.11: Results for the trajectory following experiment. Figures **(A)** and **(B)** present the $mCoP$ trajectories split up into its two dimensions x and z. Our approach enables the robot to follow the desired trajectory (gray, dash-dotted) despite moving a heavy weight with its left arm. The red dashed curves depict the trajectory of the $mCoP$ when no control action was applied and therefore shows the deviation of the $mCoP$ introduced by manipulating the weight. Figure **(C)** presents the desired figure "eight" trajectory in 2D from bird view. The gray shaded rectangles depict the contact areas of the feet with the ground. **(D)** The system reacts with sinusoidal responses in the KS-parameters to the time varying desired trajectory. In addition, we can see time varying offsets in the KS-parameters as responses to the perturbation introduced by manipulating the weight.

2.4.3 Kinematic Synergies in Single Support

In this experiment we demonstrate how to apply our approach in single support. We used two different strategies. The first strategy reused the KSs previously calculated for double support (referred to as DS-KS). We switched off the output of the control loop for the joints of the lifted leg and set the desired $mCoP$ position to the center of the reduced support polygon (defined by the single supporting foot). The second strategy was to design new KSs for single support (referred to as SS-KS). We used

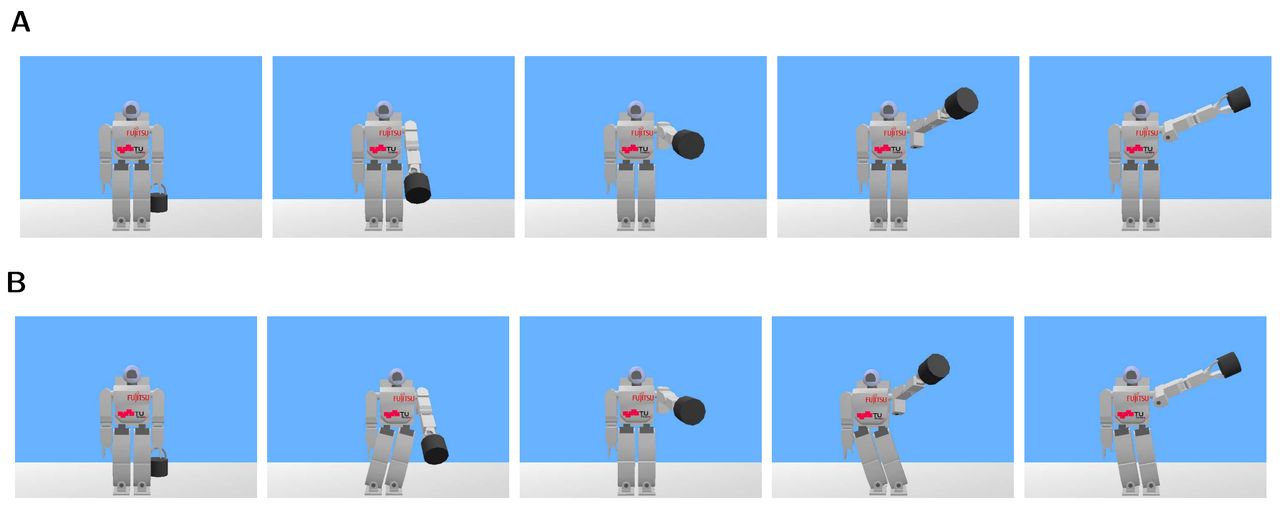


Figure 2.12: Postures of the simulated HOAP-2 for the trajectory following task. The corresponding time points from left to right are 0 (start), 12.5, 15, 17.5, and 25 (end) seconds. **(A)** In the first row no control action was applied and therefore the plain arm trajectory for manipulating the heavy weight is shown. **(B)** The second row presents the time course when the controllers were active. Note that at time point 12.5s (second screenshot from the left) the robot leaned to the left in order to follow the desired trajectory. At the end of the simulation (last screenshot on the right at 25s), the robot leaned to the right in order to compensate for the heavy weight in its left arm.

the same procedure as described previously in Subsection 2.3.1, with the distinction, that we used a different initial position (the one shown in Figure 2.13A) and we only optimized the joint angles of the supporting leg.

In the experimental setup the robot stood only on its left foot. The right foot had no contact to the ground and therefore the right leg was free to perform any desirable movement, for example, a kick motion. The initial posture can be seen in Figure 2.13A. The corresponding s-values for this posture were $s_x = 0$ and $s_z = 0.195$ for *DS-KS* and $s_x = s_z = 0$ for *SS-KS*.

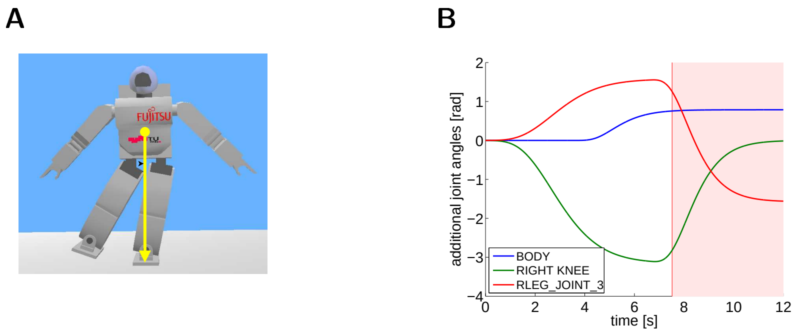


Figure 2.13: Setup for the single support task. Figure **(A)** shows the initial posture. The yellow circle denotes the CoM of the robot and the arrow points to the pCoM, which is located at the center of the support polygon. Figure **(B)** shows the joint angle trajectories, which were used for the kick motion. When no balance control was applied, the robot lost balance and tipped over at about $7.5s$ (red vertical line).

In order to demonstrate the validity of both strategies, we moved the body joint and the hip joints of the left leg (these joints were not under the control of the *KS*s) in order to perform a kick motion, which also included the upper trunk (see Figure 2.13B). For the robot this movement represented an *internal perturbation* as discussed in Subsection 2.3.3. When no balancing control was active, after about 7.5s of simulation time, the robot tipped over and fell. With the controllers switched on, the robot was able to keep balance during the kick motion (in both cases, *SS-KS* and *DS-KS*). Figure 2.14 shows the time course from 2s to 12s of this experiment with *DS-KS*. Similar results were obtained with *SS-KS*. Figures 2.15A and 2.15B show the trajectories of the *KS*-parameters s_x and s_z. Note that in the case of *DS-KS*, there was an offset at the beginning of the simulation for the *KS*-parameter s_z. This reflects the offset of the initial posture for single support from the original initial posture for double support. Figures 2.15C and 2.15D present the errors during the simulation. The controllers counteracted the disturbances correctly and kept the errors close to zero for both strategies. The dashed red curve shows the errors when no controllers were activated. Note that the scales of the y-axes of the plots in Figure 2.15 are different for the dimensions x and z.

This is a consequence of the used kick motion which mostly affected the $mCoP$ in the x direction (forward/backward). Both strategies (*DS-KS* and *SS-KS*) showed a similar performance (see Figures 2.15C and 2.15D). As a consequence, we can see that the *KS*s can also be used for different, albeit related tasks, for which, in the first place, they have not been designed for. This might also help to reduce the number of needed *KSs* in real world applications, because related tasks might share the same set of *KS*s.

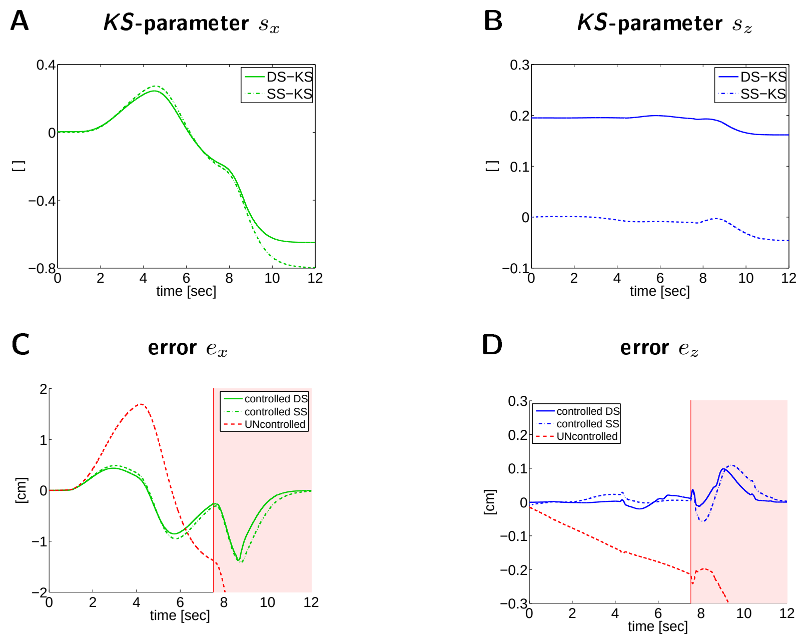


Figure 2.15: Results for the single support task. The left column shows the results for the x-dimension ($\mathbf{\Phi}_x$, forward/backward) and the right column the results for the z-dimension ($\mathbf{\Phi}_z$, left/right). The Figures **(A)** and **(B)** show the responses of the *KS*-parameter s_x and s_z for both approaches (*SS-KS* and *DS-KS*). Figures **(C)** and **(D)** show the errors. The red dashed curves denote the errors when no balance control was active. In this case the beginning of the red region indicates, when the robot tipped over and lost balance.

2.4.4 Robustness to Changes in the Model of the Robot and the Controller Parameters

The kinematic synergies are based on the static model of the robot. Since uncertainties in the model parameters (lengths and masses) are common, it is desirable to have a framework, which is robust to changes in those parameters. Moreover, such a robustness simplifies a transfer from the simulation to a real robot. In addition, it would be beneficial to have a wide range of valid control parameters, i.e., K_P and K_D, which

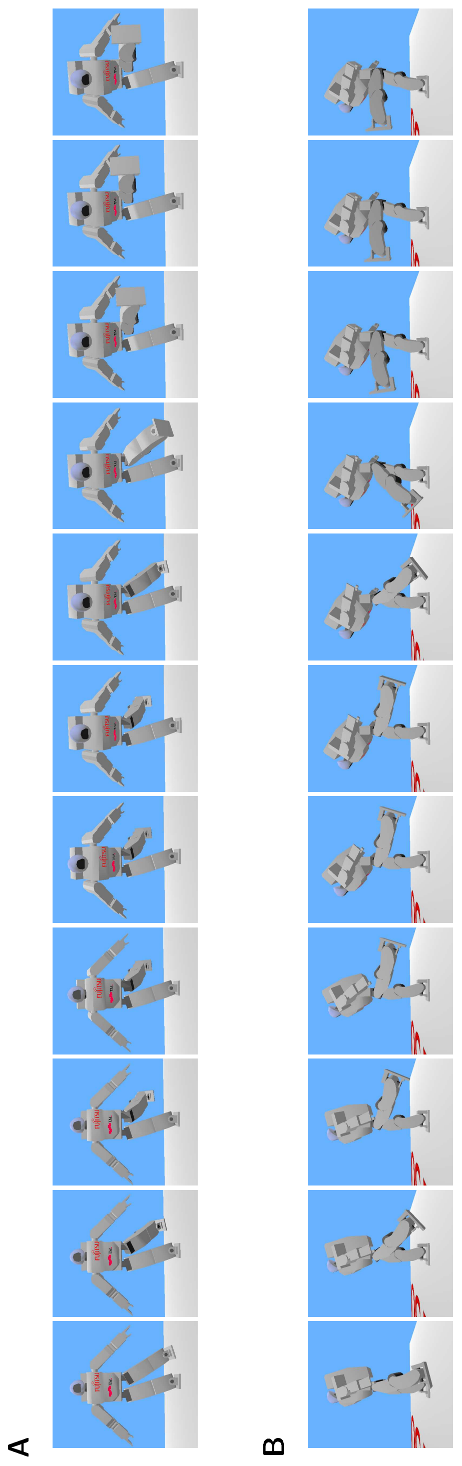
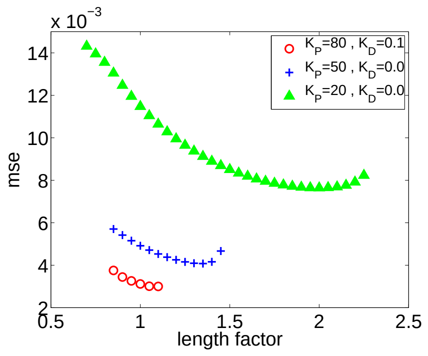


Figure 2.14: Postures of the simulated HOAP-2 for the single support task. The first row shows the robot from the front and the second row shows the robot from the side. Screenshots were taken every second from the 2nd to the 12th second.

are able to balance the robot. In the following experiments we demonstrate that our proposed setup is widely robust to variations of these parameters.

In a first experiment we varied the size of the robot by changing the length of every link by a multiplicative *length factor,* ranging from 0.5 to 2.5. We used the trajectory following task from Section 2.4.2, where the robot had to follow a figure eight trajectory with its *mCoP*, while it manipulated a heavy weight. The *KSs* were kept constant. First, we used the same controller parameters for both controllers as in the original task ($K_P = 80$ and $K_D = 0.1$). The robot was able to keep balance for a *length factor*, which ranged from 0.85 to 1.1. In Figure 2.16 the mean squared errors for the x-dimension[9] for successful *length factors* (the robot kept balance) are indicated by red circles for these controller parameters. In order to demonstrate how to improve robustness, we increased the response time of the controllers by setting the controller parameters to $K_P = 50$ and $K_D = 0.0$. In this case, successful *length factors* ranged from 0.85 to 1.45 (indicated by blue crosses in Figure 2.16). Note that the mean squared error only increased slightly. We also tested an even slower controller ($K_P = 20$ and $K_D = 0.0$), which resulted in a fairly large range from 0.7 to 2.25 (indicated by green triangles in Figure 2.16). However, the used controller was too slow to follow the desired trajectory, which can be seen in the high mean squared error values. The corresponding *mCoP* trajectories of all three controllers can be seen in the right plots of Figure 2.16. The black lines are the target trajectories. The conclusion of the experiment is that the proposed setup is robust to changes in the lengths of the robot. In addition, the results suggest that there is a tradeoff between the robustness of the approach and the response times of the controllers. Similar results were obtained, when the masses as well the lengths were changed simultaneously to simulate growing.

[9] Similar plots were obtained for the z-dimension.



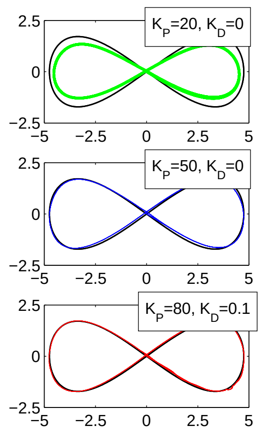


Figure 2.16: Results on robustness to changes in the **lengths**. The lengths of all links were multiplied by a *length factor*. The plot shows three different settings for the controller parameters, resulting in different response times of the controllers. The red circles show the mean squared error (mse) for the controller ($K_P = 80$ and $K_D = 0.1$) with the shortest response time. The red circles are shown for the range of successful *length factors* (the robot kept balance) from 0.85 to 1.1. By increasing the response time (controller parameters were set to $K_P = 50$ and $K_D = 0.0$.) the range (from 0.85 to 1.45) of successful *length factors* and therefore the robustness of our approach could be increased. However, also the mse increased slightly, which indicates a worse tracking performance. With an even longer response time ($K_P = 20$ and $K_D = 0.0$), the region of successful *length factors* (from 0.7 to 2.25) also grows, however, the controller was no longer able to follow the desired trajectory (indicated by the large mse values). The results point to the fact, that there is a tradeoff between the robustness of the approach and the response time of the controller.The right plots show the corresponding *mCoP* trajectories for the three controllers (at a length factor = 1).

In a second experiment we provide an evaluation of the robustness of our approach to the choice of the controller parameters. We used the single support task described in Subsection 2.4.3 and varied the K_P and K_D parameters over several decades. We evaluated which parameter settings (K_P/K_D-pairs) were successful, i.e., the robot was able to keep balance. The results can be seen in Figure 2.17. Successful parameter settings are highlighted in green. Note that the region of successful settings ranges over two decades for both parameters. This suggests that our approach is robust to the choice of the controller parameters and, thus, appropriate parameters are easily found. Moreover, this robustness potentially allows us to combine our approach with adaptive control [2] or online policy search methods [27].

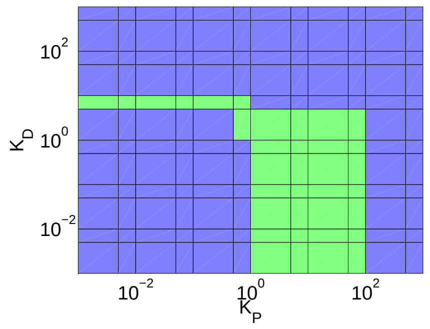


Figure 2.17: Region of evaluated controller parameters for the single support task described in 2.4.3. Successful parameter settings (for which the robot was able to keep balance) are highlighted in green. Note that the scales of the axes are logarithmic. The region of successful controller parameters ranges over two decades for both parameters, indicating that our approach is robust to the choice of the control parameters.

2.4.5 Comparison to an Online Jacobian Pseudo-Inverse Approach

We performed a comparison of our kinematic synergy setup to an online Jacobian Pseudo-Inverse (JPI) approach [48]. This approach performed online an optimization similar to the one we used for the offline construction of the *KSs*. In order to be responsive to external perturbations and model uncertainties we had to plug the JPI into a feedback control loop.

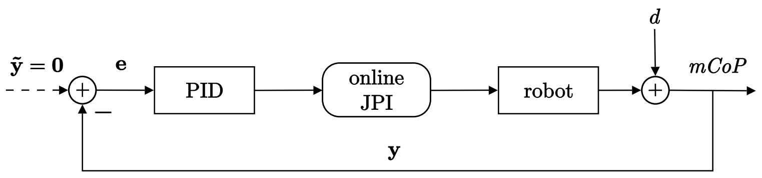


Figure 2.18: Schematic setup of the online Jacobian Pseudo-Inverse (JPI) approach, to which we compared our approach (Figure 2.8). Instead of fixed kinematic synergies this approach has to run online an optimization process (based on a JPI) at every single time step to calculate the optimal joint angles velocities.

Figure 2.18 shows the considered setup. In order to compare both approaches the robot had to track a rectangular trajectory (with rounded edges) centered at the center of the support polygon. We systematically increased the size of the rectangle and the speed of the trajectory and compared the maximum quantities, at which the robot tipped over. The differences between the two approaches for both limits (rectangle size and speed) were less than 1%. Hence, there was no significant difference in their performances. This suggests that the complex Jacobian Pseudo-Inverse computations can be performed offline (in order to construct the *KSs*) without a significant loss of performance. Note that the JPI approach needs to apply online sophisticated, time intensive calculations, while our approach is based on a much simpler control law using only a PID controller. A comparison of the online computation time of both approaches revealed a speed-up factor of 80 in favor of our approach. The results also show that the performance loss due to the linear superposition[10] of the two *KSs* is negligible for humanoid balancing.

2.4.6 Experiments with a Real HOAP-2 Robot

In our final experiment we transferred our approach to a real HOAP-2 robot. Due to the previously demonstrated robustness against model uncertainties, we were able to simply reuse the same *KSs* as in our simulations, even though the static model used for the *KSs* did not perfectly match the static model of the real robot.

We investigated two different setups. In the first setup the robot stood on the floor (denoted by **F**) and we applied external forces. This was done by applying an almost constant force from different directions for approximately 1 to 2 seconds by pushing the robot. In the second setup (denoted by **P**) we reproduced the surfboard task. The robot stood on a movable platform, which was mounted on a plastic sphere in order to resemble the surfboard with its two degrees of freedom. In contrast to the simulated experiment, no additional external forces (winds) were used (only the movement of the platform represented an external force). Note that in both setups the robot had no knowledge about the onset times, the directions or the amplitudes of the applied external forces.

The first row of Figure 2.20 shows the responses of the robot to pushes from different directions (setup **F**). The second row shows responses of the robot to different movements of the supporting platform (setup **P**). The robot counterbalanced the applied external forces in order to keep its $mCoP$ at the middle of the support polygon in each of these cases.

In Figure 2.19 we show typical *KS*-parameters and the error signals recorded while the robot was pushed from different directions (in setup **F**). Note that, except for a short time period after a change of the applied external force, the error was kept close to zero. This indicates that the robot always tried to maintain its $mCoP$ at the center of the support polygon.

[10]Note that the JPI approach does not use a linear superpositions, but rather simultaneously optimize for both output dimensions, i.e., $y \in \mathbb{R}^2$.

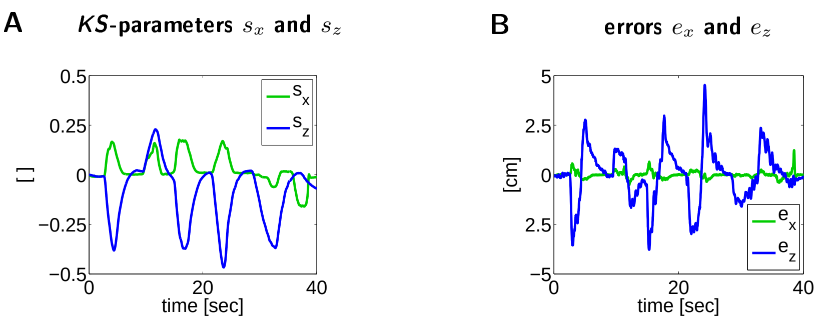


Figure 2.19: The *KS*-parameters and the errors signals recorded during an experiment with the real HOAP-2 robot. The robots was pushed from different directions (setup **F**). The left figure shows the *KS*-parameters and the right figure shows the corresponding error signals. We can see that, except for a short time period after a change of the applied external force, the error is kept close to zero. This indicates that the robot always tried to maintain its *mCoP* at the center of the support polygon.

2.5 Conclusion

We have presented a new approach towards balance control of a humanoid robot that is based on inspiration from biology. We have formalized the concept of a kinematic synergy (*KS*) that resembles the concept of a muscle synergy in physiology, and which reduces the dimensionality of the action space of the robot. We have shown that two kinematic synergies can be constructed for balance control of the humanoid robot HOAP-2 in such a way that their superposition is almost linear (like in biological paradigms), although each *KS* itself is highly nonlinear. Based on this concept we were able to demonstrate that it is possible to move the time intensive calculations of the optimization process offline and therefore keep the needed online calculations simple and fast. We have demonstrated, both through computer simulations and through experiments with the real robot HOAP-2, that this strategy makes it possible to virtually reduce the highly nonlinear balance control problem of the robot to a linear control problem (as long as the required movements are not too fast).

We showed that, in contrast to other approaches, which are based on an exact dynamic model of the robot, our proposed combination of *KSs* and linear controllers enables a humanoid robot to counterbalance unknown external forces of different kinds. Additionally, we showed that robustness to parameter changes in the model as well to changes in the controller parameters is an inherent property of the proposed approach. Based on this robustness we were able to transfer in straightforward manner this new approach for balance control from a simulated to a real HOAP-2 robot.

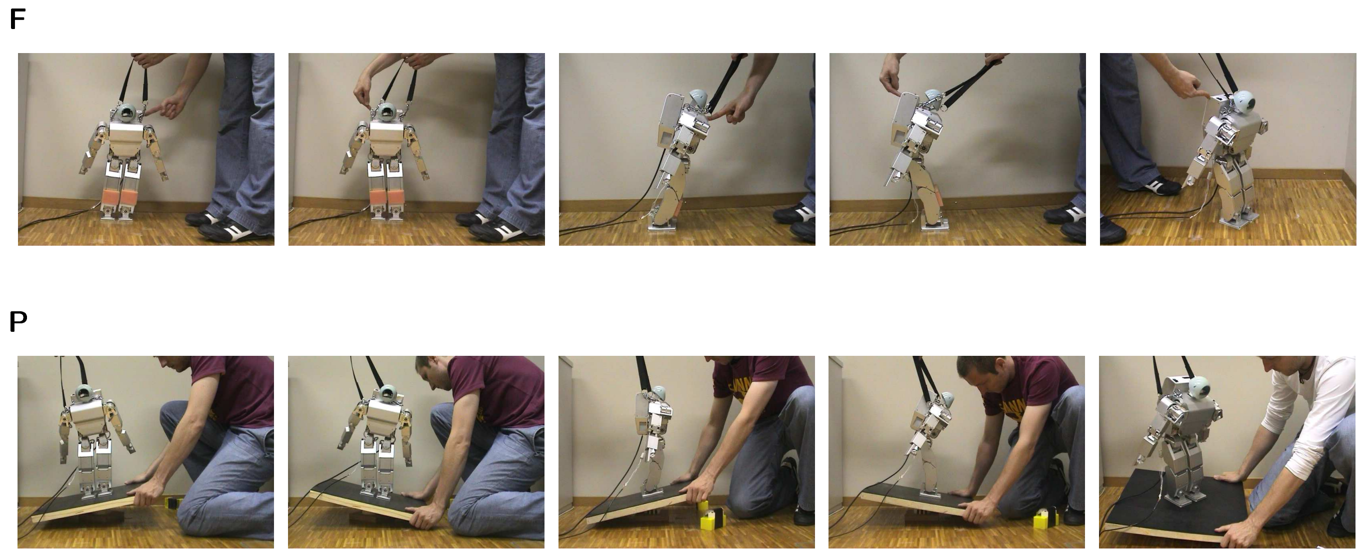


Figure 2.20: Resulting responses of the HOAP-2 to external forces. The screenshots were made during dynamic action. The top row shows screenshots for experiments while standing on the floor (setup **F**). External forces were applied by pushes. The second row shows screenshots of experiments with the robot standing on a movable platform (setup **P**). External forces were applied by moving the platform. In any of these situations the robot acted correctly and moved its $mCoP$ to the desired position at the center of the support polygon.

We expect that both, the drastic dimensionality reduction of the action space and the resulting linearization of the robot control through the use of suitable *KSs*, pave the way for future learning-based solutions to movement control problems for humanoid robots.

2.6 Acknowledgment

This chapter is based on the publications *"Biologically Inspired Kinematic Synergies Provide a New Paradigm for Balance Control of Humanoid Robots"* [16] and *"Nonlinear Kinematic Synergies Enable Linear Balance Control of a Humanoid Robot"* [17]. Both papers were jointly written by myself (HH), Gerhard Neumann (GN), Auke J. Ijspeert (AI) and Wolfgang Maass (WM). The optimization process with the Inverse Kinematics was implemented by Gerhard Neumann (GN), the simulations and the experiments with the real HOAP-2 robot were designed and implemented by HH (but the JPI-approach of Section 2.4.5, which was implemented by GN).

Chapter 3

Morphological Computation of Nonlinear, Time Invariant Filters with Fading Memory

Contents

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

3.1 Introduction

Most classical robot designs are based on rigid body parts connected by high torque servos and a central controller to coordinate them. This approach follows the view that the physical body is some complex (dynamic) system, which has to be dominated by a cleverly designed central controller. Although this is the standard approach, the resulting robots typically perform poorly compared to their biological role models. They are rather inflexible, exhibit jerky movements and tend to have a high energy consumption (see for example [9]). On the other hand the vision of morphological computation[1] proposes a radical different point of view [43]. Instead of suppressing the nonlinear dynamics of the physical body, which is the reason why classical robots are built of rigid parts, the compliant physical body could be potentially employed as a computational resource. This suggests that at least a part of the computations, which are needed during interaction, could be outsourced to the physical body itself. Hence, the body is not seen anymore as a device, which is deemed to merely drag the brain around, but rather that it is highly involved in computational tasks. As a result the remaining learning or control task and its implementation is less complex, than it would be without the aid of the physical body. There are a lot of cases of biological systems, which indicate that nature itself shares this point of view. For a number of examples and discussion we refer to [43]. Inspired by that different robots have been designed considering the idea of morphological computation. A rigorous implementation of this concept are passive walkers. The first of a series was developed by McGeer [36]. Typically, such a robot has no active controller at all. Only its passive physical structure maintains the balance in a robust fashion, while it walks down a slope. Therefore, one could argue that the computation, which is needed in order to balance the robot robustly, is "computed" by the physical body itself. A further development are passive walkers with attached (active) controllers in order to enable the robots to walk even on flat ground (for example [61]). The used controllers are remarkable simple, since most of the "work" is done by the physical body. A clever design does not only simplify the controlling task, but also the task to learn to control. For example Tedrake et. al. [52] showed that the complexity of the task to learn to walk was drastically reduced by the use of a passive walker. Due to the design of the physical structure of the robot the system was able to explore online different walking strategies without loosing balance.

Next to the two legged walking robots there exist also a number of biologically inspired robots, which mimic a range of species by simultaneously implementing the concept of morphological computation. For example the simple quadruped robot by Iida and Pfeifer [20] with a mixture of active and passive joints exhibits a surprisingly

[1]We consider here the definition "computation obtained through interaction of physical form", as suggested by [42]. However, the term is closely related to the concept of *embodiment*, which is the dynamic and reciprocal coupling among brain (control), body and environment [44].

robust behavior, although no explicit control feedback is used. Another successful implementation is the artificial fish "Wanda" [63]. It exploits the dynamics between its physical body and its environment. In the physically more complex field of flying has also been demonstrated that morphological computation can play an important role, for example, to stabilize flight (e.g., [62], [49]).

Another more abstract implementation of the idea of morphological computation are tensegrity robots [41]. These robots are built of a special combination of rigid struts and compliant strings. Already simple controllers (found by genetic algorithms) were able to induce locomotion by indirectly exploiting the dynamics of the physical body.

Despite the large body of evidence, indicating that morphology plays an important role in controlling complex bodies, so far there has been no rigorous theoretical basis for this phenomenon. As far as the authors know there has been only one attempt by Paul [42]. Her line of argumentation, based on experimental experience and thought experiments, resulted in the heuristic that a physical body with a greater amount of "dynamic coupling" (complexity) has a higher possibility of a reduced control requirement. While her statement is correct, as we see later, it is rather vague. On the other hand we will provide a rigid mathematical model to describe the computational power of physical bodies. This will enable us not only to grasp the capabilities and limitations of morphological computation, but also will give us insight of how to construct physical bodies in order to be computationally more powerful than others.
This raises the question, which type of computation is useful for biological systems and therefore for biologically inspired robots? Classical computation models, such as Turing machines, simply map a batch of input numbers in an offline computation onto output numbers. However, this type of computation is far from the needs of a robot, which should act in a real environment. It has to integrate continuously information from various continuous input streams (sensory information) and map them onto multiple output streams (motor control). Typically, such streams are mathematically encoded as functions. Computations, which map from such continuous input streams to a continuous output streams, are referred to operators or filters. In lack of a better term we will use here the expression *filter*[2], denoted by $\mathcal{F}$. In principle the computation of a filter $\mathcal{F}$ involves two nontrivial computational processes. First temporal integration of information (which is needed if the current output $y(t)$ does not depend only on the actual input $u(t)$, but also on the values $u(s)$ for some time points $s<t$), and second the nonlinear combination of such temporally integrated information.
We will provide two theoretical models, each of which is able to represent both computational processes. The considered models are depicted in Figures 3.1A and 3.1C. We will demonstrate that both of them can be implemented with the help of generic

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

physical bodies, provided that they are sufficiently complex, i.e., non-rigid and diverse. Figures 3.1B and 3.1D depict two proposed corresponding real physical implementations of these models with mass-spring systems. Note that physical bodies of biological systems as well of compliant robots can be described by such mass-spring systems. We will provide proofs that such physical realizations tend to represent the two theoretical models and therefore emulate their computational powers. Furthermore, we will present a number of simulations to support this view.

For both models we are able to demonstrate (with simulations) the contribution of the morphological structure to the computation. In the first setup (Figures 3.1A and 3.1B) the morphological structure contributes only the temporal integration. Therefore, in order to complete the computation, a *nonlinear*, but static readout has to be added. In the second setup (Figures 3.1C and 3.1D) the morphology provides both necessary computational processes (i.e., temporal integration and nonlinear combination). As a consequence only a *linear*, static readout is needed. The corresponding linear "weights" can be usually calculated by some simple, supervised algorithms, such as linear regression, but our setup also offers the potential use of some reward-based [30] or even completely unsupervised learning rules (such as Slow Feature Analysis [60]). To put it in other words the learning of complex, nonlinear dynamic filters can be reduced, through the help of the physical body (morphology), to the much simpler task of learning some static, linear weights. This perspective points to a particularly interesting feature of morphological computation, namely that it facilitates the *learning* of complex filters. Usually the learning of such filters requires nonlinear optimization procedures (such as backpropagation through time in a recurrent neural network), which often get stuck in local minima of the error-function, and which also tend to generalize not too well to new inputs[3]. However, since the morphological computation reduces this learning problem to the learning of some static weights $\mathbf{w}_{out}$, it is guaranteed that learning can not get stuck in local minima of the mean-squared error function and has arguably optimal generalization capabilities.

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

[3]On the other hand results from statistical learning theory [55] imply that the resulting generalization capability of a *linear* readout is optimal in comparison to any other learning method, where $n+1$ parameters are fitted to training data (because the Vapnik-Chervonenkis dimension of such a linear gate, which needs to be low in order to provide good generalization capability, see [6], is just $n+1$).

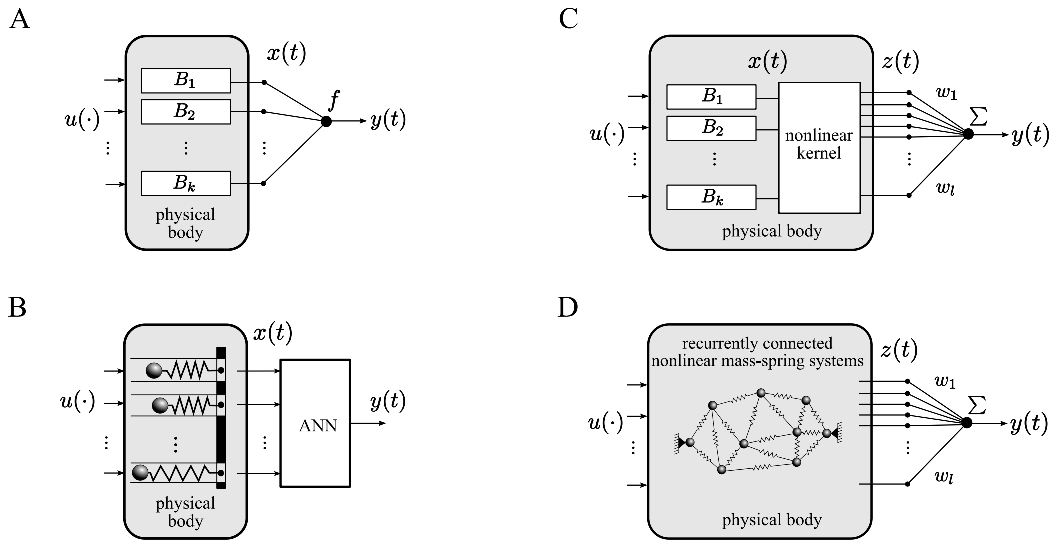


Figure 3.1: From abstract theoretical models for morphological computation to real physical bodies (consisting of mass-spring systems). **(A)** The morphology (represented here by an array of randomly chosen, time invariant, memory fading filters $B_1, \ldots, B_k$) contributes all temporal integration that is required to approximate a given filter $\mathcal{F}$. The readout f is here some memoryless, continuous function and provides the necessary nonlinear combination. Our theory provides evidence for a surprisingly large computational power of this simple architecture. **(B)** A possible implementation of (A) with a physical body. The filter array is built of an array of linear mass-spring systems and the readout is implemented by a feedforward artificial neural network (ANN). **(C)** In this architecture the morphology contributes, in addition to the temporal integration, generic nonlinear preprocessing in the form of some arbitrary kernel (i.e., nonlinear projection of $x(t)$ into a higher dimensional space). **(D)** A possible physical realization of (C). The array of filters and the kernel are both implemented by a randomly connected network of nonlinear springs and masses. In this case only a *linear* readout (instead of, e.g., an artificial neural network) has to be added externally. In the resulting computational device the output weights $[w_{out,1}, \ldots, w_{out,l}]$ are the only parameters, which are adapted in order to approximate a given complex filter $\mathcal{F}$.

In the next section we provide the theoretical foundations for morphological computations and prove that our proposed physical implementations with mass-spring systems are valid physical realizations of the theoretical models. In Sections 3.3 and 3.4 we present various simulations to support the results of the theoretical analysis. Finally, we conclude with a discussion.

3.2 Theoretical Foundations

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

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

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

Fading memory is a continuity property of filters. It requires that for any input function $u(\cdot) \in U$ the output $(\mathcal{F}u)(0)$ can be approximated by the outputs $(\mathcal{F}v)(0)$ for any other input function $v(\cdot) \in U$ that approximated $u(\cdot)$ on a sufficiently long time interval $[-T, 0]$ in the past[4]. Thus, in order to approximate $(\mathcal{F}u)(0)$, it is not necessary to know the *precise* value of the input function $u(s)$ for any time s, and it is also not necessary to have knowledge about values of $u(\cdot)$ for more than a finite time interval back into the past.

Time invariant filters are filters, which can be computed by devices that are input-driven, in the sense that the output does not depend on any absolute internal clock of the computational device. Formally one says, a filter $\mathcal{F}$ is *time invariant*, if any temporal shift of the input function $u(\cdot)$ by some amount t_0 causes a temporal shift of the output function by the same amount t_0, i.e., $(\mathcal{F}u^{t_0})(t) = (\mathcal{F}u)(t + t_0)$ for all

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

$t, t_0 \in \mathbb{R}$, where u^{t_0} is the function defined by $u^{t_0}(t) := u(t + t_0)$. Note that if the domain U of input functions $u(\cdot)$ is closed under temporal shifts, then a time invariant filter $\mathcal{F} : U \to \mathbb{R}^{\mathbb{R}}$ is characterized uniquely by the values $y(0) = (\mathcal{F}u)(0)$ of its output functions $y(\cdot)$ at time 0. In other words, in order to characterize a time invariant filter $\mathcal{F}$ we just have to observe its output values at time 0, while its input varies over all functions $u(\cdot) \in U$.

Another way to characterize nonlinear, time invariant filters with fading memory is to describe them with Volterra series[5]. A Volterra series is a finite or infinite sum (with $d = 0, 1, \ldots$) of terms of the form

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

where some integral kernel h_d is applied to products of degree d of the input stream $u(\cdot)$ at various time points $t - \tau_i$ back in the past. In order to show that such complex filters $\mathcal{F}$ can be approximated with the help of certain types of physical bodies (which consist of mass-spring systems), we use a theoretical result from Boyd and Chua [8]. This result builds on the Stone-Weierstrass approximation theorem and it implies that arbitrary time invariant filters with fading memory can be uniformly approximated by computational devices, which consist of two stages:

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

Figure 3.1A reflects this setup. A remarkable fact, which provides the basis for our theoretical analysis of morphological computation, is that the basis filters $B_1, \ldots, B_k$ of the filter bank are not required to be of a particular form. Rather, they can be chosen from any pool of time invariant, fading memory filters[6], which satisfies the following pointwise separation property.

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

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

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

This pointwise separation property is satisfied by simple, explicitly defined classes **B**, such as the class of tapped delay lines. However, it tends to be satisfied also by classes **B** of "found" physical realizations of linear and nonlinear filters. We will show that linear mass-springs systems are one type of such physically realizable filters, which form a class **B**, which has the pointwise separation property. An interesting fact is that, although no conditions are imposed on particular filters of **B**, a substantial diversity among the filters in **B** is required. An remarkable consequence is that a physical implementation of such a filter bank (in form of a morphological structure) has to exhibit this substantial diversity. While classical approaches to control robots try to avoid such complexity, or at least try to reduce it, our theoretical model of morphological computation demands it and therefore provides potentially an explanation of the complexity of biological systems[7].

Based on the definition of the pointwise separation property and on [8] (see Theorem 1 in [31] and Theorem 3.1 in [32] for a detailed proof) we can state following theorem:

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

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

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

In order to apply this theorem to real physical implementations of the proposed computational model we have to decide on how to implement the basis filters and the readout function. One possible implementation is to use real physical linear mass-spring systems to build the filter bank and an artificial neural network (ANN) as readout function. In order to show that this choice is consistent with the previously stated theorem, we have to demonstrate that linear mass-spring systems are time invariant, have fading memory and that a pool of such systems has the pointwise separation property. Regarding the readout we have to demonstrate that a pool of ANNs has the universal approximation property, which has already been proved by [19] for the case of feedforward neural networks with one hidden layer. Note that in a biological system the nonlinear readout might be implemented by a biological neural network.

[7]Note that Paul [42] came to a similar conclusion.

Now let us prove the validity of using linear mass-spring systems to build the filter bank. A single linear mass-spring system can be described by following equations

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -\frac{k}{m}x_1 - \frac{d}{m}x_2 + \frac{1}{m}u \\ y &= x_1, \end{aligned} \tag{3.1}$$

where x_1 is the displacement relative to the resting length l_0 of the spring, x_2 the rate of change of x_1 (velocity $\dot{x}_1$), $k \in \mathbb{R}^+$ the linear spring constant, $d \in \mathbb{R}^+$ the linear damping constant, m the mass of the endpoint and u the sum of all external forces acting on the mass. First, the dynamic system of Equations 3.1 is time invariant for obvious reasons. Second, we have to show that the system has fading memory. Since it is finite-dimensional and linear, it is sufficient to demonstrate that it is exponentially stable (see Section 5.1 in [8]). The eigenvalues of the system are $s_{1,2} = -d/2m \pm \sqrt{(d/2m)^2 - (k/m)}$. Since in real physical realizations of such systems $k, m \in \mathbb{R}_{>0}$, the real part $-d/2m$ is negative for any values of k and m. Hence, the system is exponentially stable and therefore has the property of fading memory. Third, the pointwise separation property of a pool of similar systems was discussed in Section 5.2 of [8]. They showed that this property holds for a special class of systems (Wiener's Laguerre systems) and mentioned that this is true not only for this special subset, but for all finite-dimensional, linear dynamic systems, to which the system of Equations 3.1 also belongs to. Hence, real physical linear mass-spring systems can be used as basis filters $B_1, \ldots, B_k$ in the setup with feedforward[8] mass-spring systems as depicted in Figure 3.1B.

Of course there exist a number of other possible implementations. A closely related morphology in a biological system is the structure of the wings of a bird. A number of diverse feathers receive the same input (i.e., air pressure) and mechanoreceptors measure the distortions. This could represent a biological implementation of the filter bank of our proposed theoretical model. Remarkably, the resulting morphological computation has already been considered in [49]. They used nonlinear angular springs to simulate the distortions of the feathers and combined it with simulated mechanoreceptors and a neural network (i.e., nonlinear readout). The network weights were found by genetic algorithms. While their design was inspired by the biological system itself, we provide here a theoretical model, which might explain their results.

So far we have only considered a setup with a clear separation of the temporal integration (implemented by a filter bank of linear mass-spring systems) and the nonlinear combination (implemented by an ANN). A biologically more plausible setup would be to dissolve this separation. In this sense a more practical application could be to choose

[8]As opposed to the *recurrent* networks as sketched in Figures 3.1C and D.

for **R** the pool of functions consisting of a fixed nonlinear *kernel.* The notion of a kernel[9] that we use here is closely related to the notion of a kernel for Support Vector Machines in machine learning [55]. However, whereas a kernel for a Support Vector Machine is a virtual mathematical concept, we are considering here concrete physical implementations of a kernel. As a consequence, such a kernel can only satisfy the kernel property for a fixed finite range. However, sufficiently large and randomly connected analog circuit of sufficiently many and diverse nonlinear components tend to map a large class of pairwise different inputs onto linear independent outputs. Therefore, a particularly tempting option for morphological computation is to let both, the filter bank and the kernel, be realized by a single physical body. We will demonstrate with the help of simulations that random, recurrently connected networks of *nonlinear* springs and masses tend to have this property. In other words, such a physical body tends to carry out temporal integration and nonlinear combination at once. Note that in contrast to the setup with feedforward mass-spring systems, where the readout was an ANN, in this case only an additional *linear* readout is required. Hence, learning to approximate a given nonlinear (time invariant, fading memory) Filter $\mathcal{F}$ is reduced to the simpler task of learning some weights, providing a number of advantages as already discussed in the introduction. Figure 3.1C depicts this idea of combing both computational processes in one physical body. Figure 3.1D depicts the corresponding proposed physical implementation as a random, recurrent network of nonlinear springs and masses.

Note that from our proposed theory it is not possible to conclude anything about the performances of the proposed morphological computation devices (not for the feedforward structure of Figure 3.1A, nor for the recurrent network of Figure 3.1C). The theory only states that sufficiently large morphological computation systems of the proposed types will provide satisfactory approximation capabilities, as long as the morphology is dynamic and sufficiently diverse. However, for a given filter $\mathcal{F}$ the theory is not able to specify how big sufficiently large is. Neither is it possible to conclude directly[10] from the theory which mass-spring systems (i.e., which physical properties) are needed for a well performing morphological computation device. This leaves us with the only option to use generic structures, i.e., to use randomly chosen mass-spring systems to construct the morphological computation devices. Naturally, this raises the questions is this approach still applicable? The answer is yes. We present a number of simulations of the proposed physical implementations (Figures 3.1B and 3.1D) applied to real world

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

[10]However, it might be possible to find well performing physical bodies for a given filter $\mathcal{F}$ by optimization schemes (e.g., with genetic algorithms).

computational tasks (which are of interest for robotics) and demonstrate that already relatively small generic structures can be used to emulate complex, nonlinear filters $\mathcal{F}$.

3.3 Morphological Computation with Feedforward Mass-Spring Systems

In this section we present simulations of the proposed physical realization of the morphological computation setup with feedforward mass-spring systems (Figure 3.1B). The simulations consisted of an array of parallel linear mass-spring systems (each of them described by Equations 3.1). All static, but nonlinear readouts were implemented as feedforward neural networks, each of one with one hidden layer of sigmoidal neurons and one linear gate as output. In the simulation we used a generic morphological structure, i.e., the values, which defined the properties of the involved mass-spring systems (i.e., spring constants k and damping constants d) were drawn randomly from a defined range. The simulations were implemented in Matlab and simulated at a time step of 1 ms.

We demonstrate that our proposed morphological computation device with feedforward mass-spring systems is in principle able to emulate a Volterra series operator. In order to have a clear, but nontrivial example we chose a Volterra series consisting of a quadratic term with a Gaussian kernel. The chosen Volterra series operator $\mathcal{V}$ is of the form

$$\begin{aligned} y(t) &= \mathcal{V}u(t) = \\ &= \iint_{\tau_1,\tau_2 \in \mathbb{R}_0^+} h_2(\tau_1,\tau_2)u(t-\tau_1)u(t-\tau_2)d\tau_1 d\tau_2 \quad , \end{aligned} \tag{3.2}$$

where $u(t)$ is the input and h_2 is a Gaussian kernel with $\mu_1 = \mu_2 = 0.1$ and $\sigma_1 = \sigma_2 = 0.05$ (in seconds), i.e., $h_2(\tau_1,\tau_2) = \exp\left((\tau_1-\mu_1)^2/2\sigma_1^2 + (\tau_2-\mu_2)^2/2\sigma_2^2\right)$, which is defined for $\tau_1,\tau_2 \in [0, 0.2]$ s. A plot of the kernel can be seen in Figure 3.2.For the simulations we used a discretized version of the kernel with a discretization step of 1 ms. Note that any computational model, which should approximate this Volterra series operator $\mathcal{V}$, must provide temporal integration (the delays τ_1 and τ_2) and nonlinearity (the quadratic term $u(t-\tau_1)u(t-\tau_2)$).

For the input we chose a product of three sinusoidal functions with different frequencies: $u(t) = \sin(2\pi f_1 t)\cdot\sin(2\pi f_2 t)\cdot\sin(2\pi f_3 t)$ with $f_1 = 2$, $f_2 = 3.1$ and $f_3 = 4.2$ Hz. The resulting signal can be seen in Figure 3.3A. The result after applying the given Volterra series operator $\mathcal{V}$ to this input signal $u(t)$ can be seen in Figure 3.3C (red line). This signal was the target output for our computational device (i.e., target).

The input signal $u(t)$ was applied to 10 linear mass-spring systems (filter bank). They all had different, random spring and damping constants. The values were randomly drawn from a log-uniform distribution from the interval [0.1, 150]. The responses of

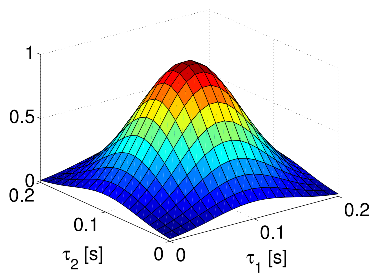


Figure 3.2: Quadratic kernel $h_2(\tau_1, \tau_2)$ used to define a Volterra series operator $\mathcal{V}$, which should be approximated by our morphological structure with feedforward mass-spring systems in combination with a nonlinear readout (i.e., as depicted in Figure 3.1B).

all linear mass-spring systems to the input can be seen in Figure 3.3B. They served as inputs to the ANN, which consisted of 20 hidden sigmoidal nodes and one linear gate as output. The weights of the ANN were adapted via BFGS quasi-Newton algorithm. For more details please refer to the supplementary material on the corresponding homepage (http://www.igi.tugraz.at/helmut/thesis). Figure 3.3C shows the performance after learning. The red line is the target signal, i.e., $\mathcal{V}u(t)$, and the blue line is the output of our morphological computational device. The achieved mean squared error (mse) was $1.06 \cdot 10^{-3}$.

In order to demonstrate the contribution of the morphological structure to the computation we compared the results to the case when no physical body (no array of mass-spring systems) was available and only the nonlinear readout (i.e., ANN) on the raw input signal remained. In order to have the same number of weights the ANN was resized accordingly. The results can be seen in Figure 3.3C. The green line is the output of the plain ANN after learning. One can see clearly that this approach failed to emulate the given Volterra series operator. The reason is that the ANN is only a static readout and is not able to represent the necessary temporal integration, which was contributed in the previous case by the morphological structure. As already argued in the introduction the setup offers the possibility of multiplexing, i.e., the same fixed[11] morphological structure can potentially be used for a number of different tasks. Note that the ability of multiplexing is a beneficial feature, since the morphological structures of real robots (and biological systems) are to a high degree fixed[12]. In order to demonstrate multiplexing we used the same morphological structure (same filter bank) and the same input of the previous task and applied it to a new task by simply adding a new readout.

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

[12]However, for the biological case exists experimental evidence, that the stiffness can change in order to adapt to different environments (e.g., [12]).

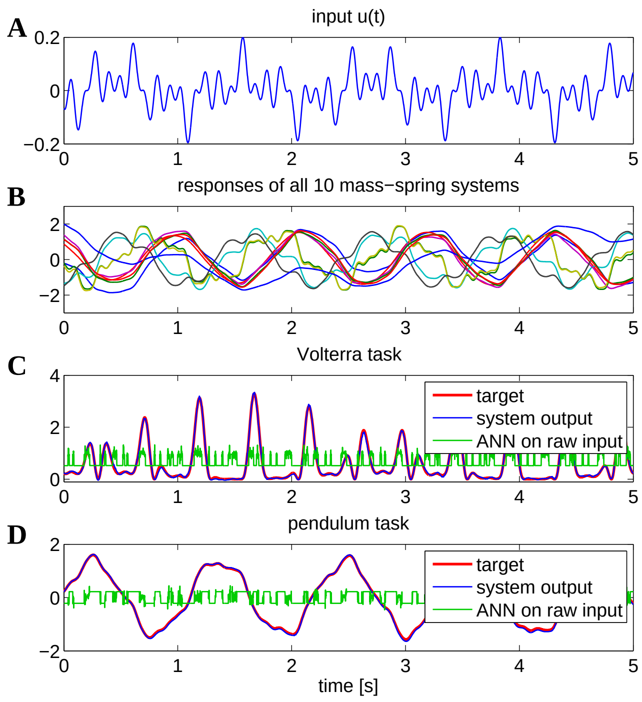


Figure 3.3: Applying a feedforward morphological computation device to approximate the Volterra series operator $\mathcal{V}$ (defined by Equation 3.2) and the pendulum (Equation 3.3) simultaneously with one morphological structure (i.e., multiplexing). **(A)** The used input signal $u(t)$, which consisted of a product of three different sinusoidal functions ($f_1 = 2$, $f_2 = 3.1$ and $f_3 = 4.2$ Hz). **(B)** the responses of all 10 mass-spring systems to this input (for a better readability the outputs were normalized to zero mean and a standard deviation of one). **(C)** the performance of the proposed morphological computation device for the Volterra task. The red line is the target (applying the Volterra series operator to the input, i.e., $\mathcal{V}u(t)$) and the blue line shows the output of the morphological computation device. The green line shows the performance of the device, when no morphological structure was available, i.e., only the nonlinear readout of the ANN was applied to the raw input data. Clearly this approach fails, since the ANN is only a static readout and is not able to represent the necessary temporal integration, which was contributed in the previous case by the morphological structure. **(D)** The pendulum task: The red line is the target, the blue line the output of the morphological computation device and the green line, when no morphological structure was available.

For the additional task we chose from an interesting subclass of nonlinear filters, which can be described by Volterra series, namely the class of nonlinear dynamical system with fading memory[13]. An example of such a dynamical system is the damped pendulum, which can be described by following equations (taken from [26])

$$\begin{aligned}\dot{\alpha} &= \omega \\ \dot{\omega} &= -\frac{g}{l}\sin(\alpha) - \frac{\mu}{m}\omega + \frac{1}{ml^2}A\tau \\ y &= \alpha,\end{aligned} \tag{3.3}$$

where α is the angle, ω the angular velocity, $g = 9.81$ m/s^2 the gravitational acceleration, l the length, m the mass of the bob and μ the friction coefficient. The constant A is a proportional factor, which was set to $A = 40$ in order to drive the system into the nonlinear domain of the state space. For the same reason we set in the simulations $l = 0.5$, $m = 0.1$ and $d = 1$. The input to the system was the torque τ and the output was the angle α. In order to obtain suitable targets we simulated system 3.3 at a time step of 1 ms with Matlab's internal ordinary differential equation solver. The input $u(t)$, now interpreted as torque $\tau(t)$, was the same as in the previous task (Figure 3.3A). The red line in Figure 3.3D shows the corresponding output (i.e., target).

Since we used the same morphological structure (same filter bank array) and the same input $u(t)$, consequently, the responses of the mass-spring systems were the same as before (Figure 3.3B). Based on these responses as inputs an ANN with 10 hidden sigmoidal neurons and one linear output gate was trained (with the BFGS quasi-Newton algorithm) to approximate the desired targets. The performance can be see in Figure 3.3D. The resulting mse was $1.09 \cdot 10^{-3}$. As we can see the fixed generic morphological structure in conjunction with two different readouts was able to represent the two different nonlinear filters.

Again, in order to show explicitly the contribution of the morphological structure to the computation we compared the results to the case when no physical body (array of mass-spring systems) was available and only the ANN remained. The performance can be seen in Figure 3.3D, where the green line represents the output of the ANN. As before, the ANN applied to the raw input stream failed to represent the desired nonlinear filter (i.e., the pendulum equations)..

As previously argued the morphological structure has to be diverse in order to be computationally powerful. In order to show that this is true we set the properties of all the mass-spring systems in the filter bank to the same values (k and d were the same). The resulting mse for the Volterra task was 0.7937, which was more than 700 times higher than with the previously used heterogeneous filterbank.

[13]Boyd and Chua [8] have pointed out that fading memory for (time-invariant) dynamical systems is related to the *unique steady-state property*. For a discussion we refer to [7].

3.4 Morphological Computation with Recurrent Networks of Nonlinear Springs and Masses

In the previous simulations we used the approach with a strict feedforward structure (Figure 3.1A). It implemented a spacial separation between a linear but dynamic part (implemented as an array of linear mass-spring systems), which provided temporal integration, and the nonlinear, static readout (implemented as an ANN), which provided the nonlinearity. However, as we have already argued in Section 3.2, there could exist physical realizations which have the property to combine both computational aspects in a single body. We will demonstrate in the following simulations, that random, recurrent networks of nonlinear springs and masses tend to be such physical realizations. A particular interesting property of this setup is that in contrast to the setup with feedforward mass-spring systems, where a nonlinear readout (e.g., ANN) was needed, in this case only a simple *linear* readout has to be added in order to *complete* the morphological computation (compare Figures 3.1B and 3.1D).

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

3.4.1 Implementation of Recurrent Networks of Nonlinear Springs and Masses

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

3.4.1.1 Constructing Mass-Spring Networks

The construction of the mass-spring networks was based on following two principles. First, the final network should be realizable as a real physical system, and second, it should be generic, i.e., not be constructed for any specific task.
A chosen number of N nodes (mass points) were randomly positioned (uniformly distributed) within a defined range of a two dimensional plane. Subsequently, we connected these mass points by nonlinear springs. In order to find reasonable, non-crossing spring connections we calculated a Delaunay triangulation on this set of points, resulting in L non-crossing spring connections. A schematic example of such a mass-spring network can be seen in Figure 4.2. Every single nonlinear spring of such a network can be described by following nonlinear dynamic system

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= -p(x_1) - q(x_2) + u, \end{aligned} \tag{3.4}$$

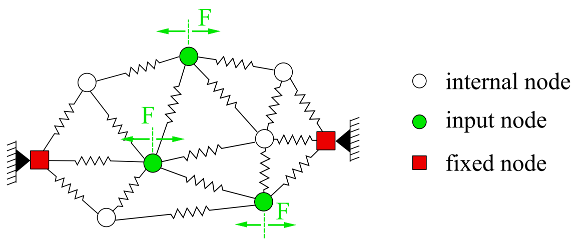


Figure 3.4: Schematic example of a generic mass-spring network. The nodes (masses) are connected by nonlinear springs. The red nodes are fixed in order to hold the network in place. The green nodes are randomly chosen inputs nodes, which receive the input in form of horizontal forces scaled by randomly initiated weights.

where $x_1 = l - l_0$ is the difference between the actual length l and the resting length l_0, $x_2 \in \mathbb{R}$ is the rate of change of x_1 (velocity $\dot{x}_1$) and u the sum of all external forces acting on it. At the beginning of the simulation we assumed the mass-spring network to be at rest (i.e., all springs were at their point of equilibrium $\mathbf{x} = [0, 0]^T$ 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, had to be monotonically increasing and fulfill $p(0) = 0$ and $q(0) = 0$[14]. Typically nonlinear springs are modeled by 3rd order polynomials [40]. Therefore, we implemented the nonlinear functions as $p(x_1) = k_3 x_1^3 + k_1 x_1$ and $q(x_2) = d_3 x_2^3 + d_1 x_2$, where $k_1, d_1 \in \mathbb{R}_{>0}$ and $k_3, d_3 \in \mathbb{R}^+$ defined the properties of the spring. In order to get a rich kernel, as argued in Section 3.2, 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. The left most and the right most mass nodes were fixed in order to keep the network in place (red squares in Figure 4.2). A certain percentage of all nodes were randomly chosen to be input nodes (green nodes in Figure 4.2). During simulation they received a linearly scaled version of the current input in form of a horizontal force. Before the simulation started the input scaling factors (weights $\mathbf{w}_{in} = [w_{in,1}, w_{in,2}, \ldots]^T$) had been randomly drawn from a certain range and had been fixed subsequently.
The linear readout of the network was defined as the weighted sum of all actual spring lengths $y(t) := \sum_{i=1}^{L} w_{out,i} l_i(t)$. The output weights ($\mathbf{w}_{out} = [w_{out,1}, w_{out,2}, \ldots, w_{out,L}]^T$), in contrast the rest of the network parameters, were adapted in the learning process.

[14]A proof for that is based on the Lyapunov function $V(\mathbf{x}) = \int_0^{x_1} p(\zeta) d\zeta + \frac{1}{2} x_2^2$, its derivative $\dot{V}(\mathbf{x}) = -x_2 q(x_2)$ and the use of a corollary of La Salle's Theorem (see Theorem 4.4 and Corollary 4.2 in [26]).

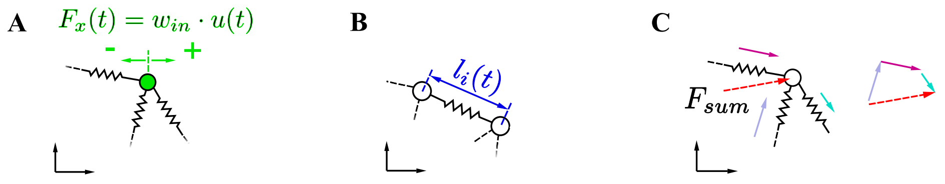


Figure 3.5: Implementation of input, linear readout and simulation of forces of the mass-spring networks. **(A)** The input is applied to an input node as a horizontal force F_x proportional to the input signal u (scaled by a randomly initialized weight w_{in} for this input node). **(B)** The readout from the network is the weighted sum of all L spring lengths $y(t) = \sum_{i=1}^{L} w_{out,i} l_i(t)$. In general the input 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.

3.4.1.2 Simulating Mass-Spring Networks

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

$$m\ddot{p}_x = F_x + w_{in}u \tag{3.5}$$

$$m\ddot{p}_y = F_y, \tag{3.6}$$

where $\ddot{p}_x$ and $\ddot{p}_y$ were the accelerations of the mass point relative to a global reference frame split up into its two spacial dimensions, F_x and F_y were the forces acting on the mass in the corresponding spacial dimensions, and $w_{in}u$ was the weighted input. Note that the input was defined as a horizontal force (see Figure 3.5A) and if the mass point was no input node $w_{in}u := 0$. For the sake of simplicity[15] 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 Equation 3.4. The value of x_1 was calculated by the actual length $l(t)$ (Euclidean distance between the two masses which the spring connected) and the resting length l_0. The velocity x_2 was approximated by $(x_1(t) - x_1(t - \Delta t)) / \Delta t$ with a time step of $\Delta t = 1$ ms. The resulting forces were calculated by the nonlinear functions $p(x_1)$ and $q(x_2)$. This procedure was repeated for all springs connected to the mass. We assumed that these forces acted along their corresponding spring axes. Finally, all spring forces acting on the regarding mass node were summed up vectorially (see Figure 3.5C). Subsequently, the resulting force F_{sum} was split up into its two spacial dimensions and added as forces F_x and F_y to Equations

[15]Note that the masses are only linear scaling factors and, since the properties of the springs were randomly drawn, could be set to 1 for all masses without the loss of generality. Nevertheless, in a real biological body (or robot) a diversity of masses is natural and it contributes further diversity.

3.5 and 3.6. If the mass point was an input node the current input $u(t)$ was added in form of a scaled horizontal force (see Equation 3.5 and Figure 3.5A). The new position and velocity of the mass were found by numerically integrating Equations 3.5 and 3.6 with the 4th order Runge-Kutta method. The same procedure was repeated for all masses. At the end of the simulation step the current output was calculated by a linear combination of the actual lengths of all springs, i.e., $y(t) = \sum_{i=1}^{L} w_{out,i} l_i(t)$ (see Figure 3.5B).

3.4.1.3 Learning the Linear Readout of Mass-Spring Networks

The structure of the mass-spring networks, as well as the parameters, which defined the physical behavior, were randomly initialized and subsequently fixed. Only the linear readout was adapted during the learning process, i.e., the weights $\mathbf{w}_{out} = [w_{out,1}, w_{out,2}, \ldots, w_{out,L}]^T$ were adapted. For learning we considered a networks of N nodes connected by L springs. During the simulation we collected the current lengths of every single spring $l_i(t)$ for $i = 1, \ldots, L$ at every time step $t = 1, \ldots, M$ in a $L \times M$ matrix $\mathbf{L}$. We dismissed data from an initial period of time (washout time) to get rid of initial transients. The target signal was also collected over time in a matrix $\mathbf{T}$. Finally, the optimal values for the output weights were calculated by $\mathbf{w}_{out}^* = \mathbf{L}^\dagger \mathbf{T}$, with $\mathbf{L}^\dagger$ being the (Moore–Penrose) pseudoinverse, since in general $\mathbf{L}$ was not a square matrix. Note that the same procedure can be applied in the case of multiple inputs and/or multiple outputs.

3.4.2 Representing Inverse Dynamics by a Recurrent Mass-Spring Network

As a first task we will demonstrate that a generic mass-spring network can be used to emulate the dynamic mapping from a trajectory of an end-effector of a robot arm in Cartesian space to the corresponding torques. We used a full dynamic two link robot arm [50], which was assumed to move in a horizontal plane. Hence, the gravitational forces could be ignored. We refer to the supplementary material[16] for further details on the robot model. Figure 3.6 shows the setup of the task.

The end-effector of the robot arm had to move along the blue trajectory. The corresponding trajectories in Cartesian space, i.e., x and y positions, are plotted in Figure 3.8A. The corresponding targets torques, which allowed the robot arm to move along these trajectories, can be seen in Figure 3.8C (red lines). These torques were found by the following process: We chose an arbitrary starting posture. Based on the x- and y-trajectories (which defined the figure eight trajectory in Cartesian space) and the Jacobian of the robot arm we calculated the corresponding trajectories of the joint angles.

[16]http://www.igi.tugraz.at/helmut/thesis

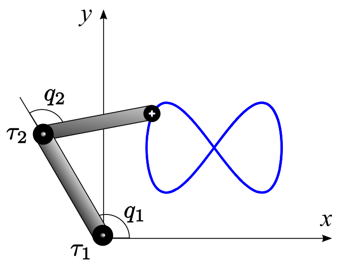


Figure 3.6: Setup for the robot arm task. The blue line is the desired trajectory for the end-effector.

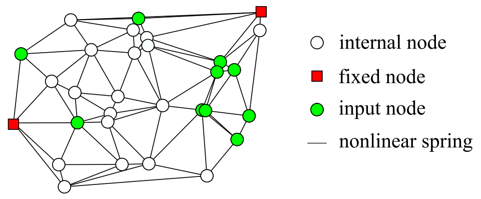


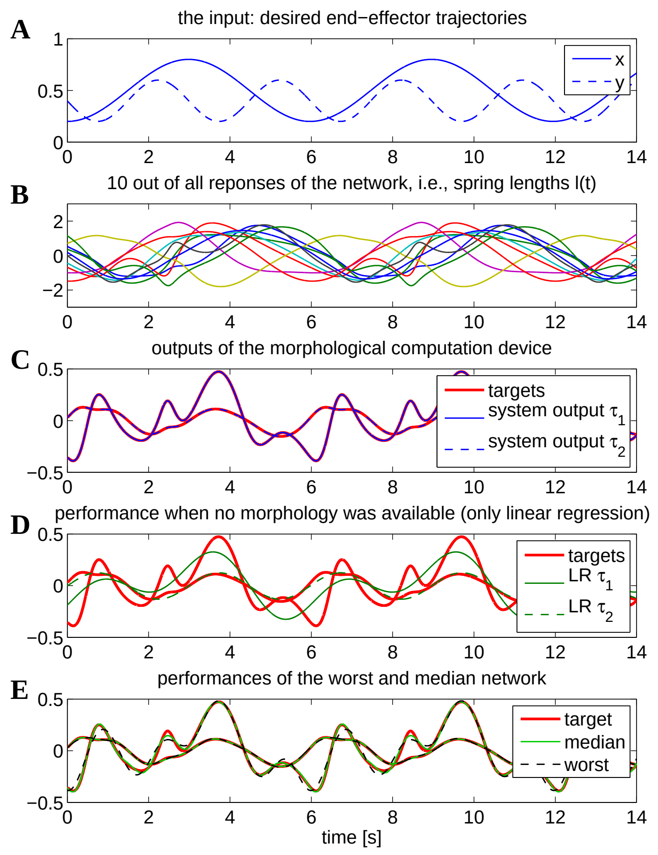
Figure 3.7: Generic mass-spring network used for the robot arm task and subsequently for the multiplexing task in Section 3.4.3. The red nodes are globally fixed and the green nodes are the randomly chosen input nodes. The network consisted of 30 masses and 78 nonlinear springs.

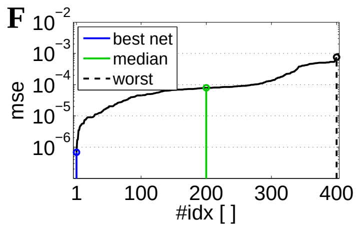
Subsequently, the corresponding torques were found by the use of PD-controllers[17] in order to follow those joint angle trajectories.
We constructed a generic mass-spring network based on the previously described process (Section 3.4.1.1). The parameters of the springs (i.e., k_1, k_3, d_1 and d_3 as defined in 3.4.1.1) were randomly drawn from the range [1, 100] (log-uniform distribution) for the values k_1 and d_1, and from [100, 200] (uniform distribution) for the values k_3 and d_3. We chose randomly 20% of all nodes to be input nodes for the first input (i.e., input signal x) and also 20% of all nodes for the second input (i.e., y). For more details please refer to the supplementary material (http://www.igi.tugraz.at/helmut/thesis). One of the obtained mass-spring networks can be seen in Figure 3.7. It consisted of 30 masses and 78 nonlinear springs.
As described in Section 3.4 the randomly chosen input nodes (green nodes) received a scaled horizontal force proportional to the input. The scaling weights $\mathbf{w}_{in}$ were randomly (uniform distribution) drawn from $[-1, +1]$. The mass-spring network responded to this inputs by changing the mass positions and the spring lengths. Figure 3.8B shows 10 typical spring length trajectories (out of all 78). For a better readability the trajectories in this plot were normalized to zero mean and a standard deviation of one. Based

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

on the targets, all 78 spring length trajectories and the previously described learning process we calculated the optimal output weights. Note that these weights were static, i.e., did not provide temporal integration, and that the resulting readout was linear, i.e., it did not provide any nonlinearity. Figure 3.8C shows the performance after learning (using the network of Figure 3.7). The red lines are the target torques and the blue lines are the outputs of the morphological computation device (solid blue for τ_1 and dashed blue for τ_2). We can see that the setup was able to represent the dynamic mapping from the Cartesian space to the robot arm torques.
In order to demonstrate the contribution of the morphological structure to the computation we compared the results to the case when no physical body (i.e., no mass-spring network) was available and only the linear readout remained. In order to do so we applied linear regression (LR) on the raw input signals. Therefore, we defined the output at time t by $\tau_1^{LR}(t) = w_1 x(t) + w_2 y(t) + w_{bias}$, where x and y were the inputs (as in Figure 3.8A) and $\mathbf{w}^{LR} = [w_1, w_2, w_{bias}]^T$ were some static weights, which were found by standard linear regression. Accordingly, we calculated the three corresponding weights for the second output τ_2. Figure 3.8D shows the performance of this approach. The red lines are the targets and the green lines are the outputs. The approach failed because it was no able to represent the necessary temporal integration and nonlinear combination. In the previous case (with the physical body) the morphological structure provided both of these computational aspects[18].
The network of Figure 3.7 was chosen based on the fact that it was the best performing network out of a number of networks constructed with the same probability distribution, i.e., the same construction parameters, which defined the ranges for the random values used for the construction process. More specifically, these construction parameters were the defined ranges, from which the spring parameters were drawn from, the percentage of all nodes, which received an input, the range for the input weights $\mathbf{w}_{in}$ and the size of the area in which we randomly placed all mass points. This raises the question, is it easy to find such a set of parameters, which defines a pool of well performing networks? Note that for example the range of possible values for the spring parameters k_1 and d_1 went over two decades ([1, 100] - see description above). This points to the fact that no tedious parameter tuning was necessary. In order to demonstrate that the used (rather broad ranged) construction parameters defined a whole set of well performing networks, and the presented network was not just a statistical outlier, we constructed 400 random networks based on exactly these parameters. Subsequently, we sorted the networks accordingly to their performances (i.e., by their averaged mean squared error over both outputs; denoted here by *mse*). The results are presented in Figure 3.8F. We can see that even the worst performing network had still a *mse* smaller than 10^{-3}. Out of the 400 mass-spring networks we chose

[18]Note that in the setup used in the Section 3.3 the physical body only provided the temporal integration.





	mse
best	$6.83 \cdot 10^{-7}$
median	$7.94 \cdot 10^{-5}$
worst	$7.59 \cdot 10^{-4}$

Figure 3.8: Representation of the inverse dynamics of a robot arm with the help of morphological computation. **(A)** The desired end-effector trajectory split up in its two Cartesian coordinates x and y (i.e., inputs). **(B)** 10 typical responses (out of all 78) of the mass-spring network to this input. For a better readability each signal was normalized to zero mean and a standard deviation of one. **(C)** The performance of the morphological computation device. The red lines are the target torque trajectories and the blue lines are the outputs of the computational device. **(D)** The performance when no morphological structure was available, i.e., only a linear regression (LR) on the actual values of the inputs remained. This approach failed to represent the dynamic and nonlinear mapping. **(E)** and **(F)** Based on the same construction parameters we randomly generated 400 networks and sorted them by their mean squared error (mse) over its two outputs. The table shows the performances of the best, the worst and the median network. The best network was used for the plot of Figure C. The performances of the worst (black dotted line) and the median network (green) are presented in Figure F.

the best network (blue line), the worst network (black dotted line) and the median network (green line)[19]. The table in Figure 3.8 lists the *mse* of them. Figure 3.8C shows the performance of the best network and Figure 3.8E the performances of the worst network (black dotted line) and the median network (green line). Similar results can be obtained for other tasks and construction parameters. This suggests that in general no tedious parameter search has to be done in order to find probability distributions to define a successful pool of networks. Additionally, this means that the physical body does not have to be tuned for a specific task in order to be a valid computational resource, as long as it is sufficiently complex and diverse. Therefore, the same morphological structure could be potentially used for a number of different tasks simultaneously (i.e., multiplexing).

3.4.3 Multiplexing Property of a Mass-Spring Network

In this section we demonstrate that mass-spring networks have the desired property of multiplexing. Note that in contrast to the multiplexing in the setup with feedforward mass-spring systems (Section 3.3), where we used different ANNs as readouts, in the case of mass-spring networks only a corresponding number of *linear* readouts is sufficient. For the following simulations we used therefore one generic network, one input and three different linear readouts to emulate three different nonlinear filters.
For the first filter we chose the previously defined Volterra series operator $\mathcal{V}$ (Equation 3.2). The second task was to emulate following 2nd order nonlinear dynamic system

$$y[k+1] = 0.4y[k] + 0.4y[k]y[k-1] + 0.6u^3[k] + 0.1 \ , \tag{3.7}$$

where $u[k]$ was the input and $y[k]$ the output at time step k. The third task was to emulate following nonlinear 10th order system

$$\begin{aligned} y[k+1] &= 0.3y[k] + 0.05y[k]\left(\sum_{i=0}^{9} y[k-i]\right) \\ &\quad +1.5u[k-9]u[k] + 0.1 \ . \end{aligned} \tag{3.8}$$

Again $u[k]$ was the input and $y[k]$ was the output at time step k. The systems 3.7 and 3.8 were both taken from [3], where they were used in order to demonstrate the performance of a new learning algorithm for recurrent networks. Note that nonlinear systems of the type of Equations 3.8 are typically hard to emulate for recurrent networks due to their long-term time dependencies [18]. Note also that our proposed morphological computation device is an analog device, which is able to map continuous input streams onto continuous output streams. However, in the simulation of this analog device we were

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

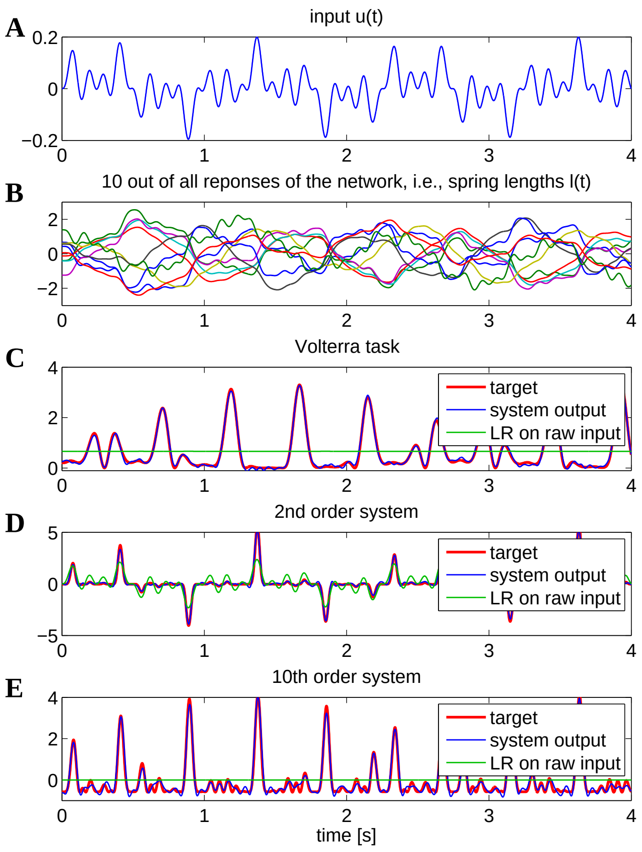


Figure 3.9: Simultaneous morphological computation of the three nonlinear filters with one generic recurrent mass-spring network (i.e., multiplexing). **(A)** The input $u(t)$, which consisted of a product of 3 sinusoidal functions. **(B)** The trajectories of 10 typical (out of 78) individual spring lengths $l(t)$ as responses to this input. **(C)** The performance for the Volterra task. The red line is the target function and the blue line is the output of the morphological computation device. The green line depicts the outputs of the device, when no morphological structure was available, i.e., only the linear readout was applied to the raw input data. Note that the result is simply a scaled version of the input with some offset. **(D)** Performance of emulating system 3.7. **(E)** Performance for the filter defined by the system 3.8.

restricted to discrete time. The simulation time step and the time step of Equations 3.7 and 3.8 were the same. Note that a real physical (analog) implementation of the morphological computation device would emulate the underlying continuous dynamic systems, which correspond to the discrete Equations 3.7 and 3.8 and which minimize the errors at the discretization time steps.

We used the same network as in the previous robot arm task (Figure 3.7). All previously chosen input nodes (in the robot task assigned for two inputs) were now defined to receive the single input $u[k]$. As input we used the same signal as previously for the experiment in Section 3.3, where we used the morphological computation device with feedforward mass-spring systems. It was a product of three sinusoidal functions and it is shown again in Figure 3.9A. Figure 3.9B shows 10 typical trajectory (out of all 78) of the spring lengths as responses of the mass-spring network to this input. The output weights for the linear readouts were found as previously described. Figure 3.9C shows the performance of our morphological computation device for the Volterra task. The red line is the target and the blue line is the output of the morphological computation device. The mass-spring network with an additional linear readout is able to emulate the nonlinear filter defined by the Volterra series operator $\mathcal{V}$. Note that, unlike to the previous Volterra task of Subsection 3.3 (with a filter bank), here the physical body (mass-spring network) provided not only the temporal integration but also the nonlinearity. Hence, in order to learn to emulate the given nonlinear filter $\mathcal{V}$, due to the use of the nonlinear and dynamic morphological structure (as a computational resource), we only had to calculate a simple linear regression.

In order to show the explicit contribution of the morphological structure to the computation we compared the results to the case when no physical body (no mass-spring network) was available and only linear regression on the raw input data remained. We used a linear regression with two weights, w_1 for the actual input $u(t)$ and w_2 to learn a bias. Hence, the resulting output at time step t was $y^{LR}(t) = w_1 u(t) + w_2$. The performance can be seen in 3.7C. The output $y^{LR}(t)$ is depicted by the green line, which is simply a scaled version of the input (with a very small amplitude) with an additional offset. Not surprisingly, pure linear regression on the raw input stream failed to represent the nonlinear filter $\mathcal{V}$, since all the required temporal integration and nonlinearity was contributed before by the physical body (mass-spring network).

Figure 3.9D and 3.9E show the performances of the morphological computation device in order to emulate the nonlinear systems 3.7 and 3.8 using the same morphological structure (mass-spring network of Figure 3.7). Again, the red lines are the targets and the blue lines the outputs of the device. The green lines depict the results, when no morphological structure was available and only pure linear regression was applied to the input stream. One can see, that also in these cases, the linear regression, which was static and linear, failed to represent the necessary dynamics and nonlinearity.

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

3.5 Discussion

Based on rigorous mathematics we introduced two theoretical models, which provide a potential explanation for the computational power of physical bodies. In order to demonstrate the applicability of the theory to real world tasks we presented a number of experiments, where we simulated physical implementations of the proposed morphological computation devices.
The proposed setups are formed by a dynamic morphological structure (i.e., the physical body with fixed parameters) and a static readout (which can be adapted). As we have shown the readout can be even linear, if the morphological structure is sufficiently *rich*. Remarkably, such simple devices are in principle able to emulate any nonlinear, time-invariant filter with fading memory by adapting a simple, linear readout. Hence, the complex task of learning to emulate such complex filters can, due to the help of the morphological structure (i.e., due to morphological computation), be reduced to the task of finding some linear weights. This suggests that physical bodies are potentially able to boost the expression power of attached linear learning systems. Note that a linear readout increases the speed of learning drastically. Furthermore it guaranties that the optimization process does not get stuck in a local minimum and it has optimal generalization capabilities.
A remarkable fact of the proposed theory is that it suggests that morphological structures, in order to be computationally powerful, should be diverse in their parameters and that they should exhibit high dimensionality. Note that both aspects are typical properties of compliant, biological body parts. However, in classical robot design these attributes are suppressed (by high torque servos and rigid body parts) in order to have a more tractable model and an easier controllable robot. Our results point to the fact that the consideration of these dynamic features are essential in order to be able to outsource computational tasks to the morphological structure and therefore simplify the control of the robot. This perspective suggests that the development of novel high-dimensional readouts from artificial limbs (e.g., acceleration sensors at many locations inside the robot) is a possible way to exploit the morphological structure. The morphological structures of biological systems might even be more suitable for this task since they provide naturally a high number of internal sensory signals and a variety of interconnected dynamic structures (muscle-skeleton system, etc.).
Another interesting aspect of the approach is that compliant physical bodies provide the necessary nonlinearities and the temporal integration for *free*. The physical structure simple reacts on its *inputs*. Actually, it is not even necessary to have real physical interpretation of all the available internal signals in order to exploit them for morpho-

logical computation.
Another remarkably property of the proposed morphological computation devices is the one of multiplexing. One morphological structure is able to provide the necessary signals for a number of nonlinear filters - only a corresponding number of readouts has to be added. While this multiplexing ability is obviously beneficial, since the physical bodies of biological system as well as of robots are to high degree fixed, on the other hand one would also assume that the computational tasks for the physical body are limited to a set of particular filters. This suggests that physical bodies or different parts of the body could be optimized regarding to their computational tasks. The resulting structures would be inhomogeneous and asymmetric as opposed to the examples presented here. This points to the need for a new type of *computational* material science and *computational* robotics, where the geometrical and statistical properties of the fine structure of different materials are analyzed (and optimized) with regard to their suitability to support through morphological computation the computations of a particular range of filters, e.g., filters that are needed to control a robot for a particular range of tasks.
Obviously these considerations will also open new perspectives for our understanding of the shape and structural properties of the body of biological organisms and consequently will lead to new types of biologically inspired robots.

3.6 Acknowledgment

This chapter is based on a draft of a paper with the tentative title *"A Theoretical Foundation for Morphological Computation"*. The authors are myself (HH), Rolf Pfeifer (RP), Rudolf M. Füchslin (RF), Auke J. Ijspeert (AI) and Wolfgang Maass (WM). All of them contributed by giving feedback and by proposing experiments. The basic impulse came from WM and AI. The writing, the proofs, the implementation of the simulations as well the design of the experiments were done my myself (HH).

Chapter 4

Morphological Computation with External Feedback

Contents

In the previous chapter we presented two theoretical models, which provided an explanation for the computational power of certain types of physical bodies. We showed that the proposed morphological computation devices are theoretically able to emulate any non-linear, time-invariant filter with fading memory. While this class of filters is very rich and it includes a wide range of relevant computations, it still has its limitations. For example, it does not include limit cycles, nor the switching between different states, since such computations request persistent memory. However, such types of computations are still of interest for robots, for example, to induce locomotion, or to carry out any other rhythmic movement. It would be also of value to be able to switch between different types of locomotion (i.e., gaits) depending on some sensory information. A way to overcome the limitations of the previous approach, and therefore to be able to emulate the mentioned computations with persistent memory, is to add external feedback. However, this requires a new type of computational model, which we will introduce in this chapter. Based on this new theory we will demonstrate that physical bodies enhanced by simple, linear feedbacks and linear readouts can be enabled to emulate virtually any nonlinear dynamic systems, as long as the physical body is sufficient diverse and dynamic (i.e.,

complex and compliant). Hence, the task of emulating complex, nonlinear dynamic systems (even with persistent memory) can be reduced, due to the help of the morphological structure, to the much simpler task of finding some linear output weights. We present a number of simulations to demonstrate that already small networks of nonlinear springs and masses (of the same type as used in the previous chapter) with additional external, linear feedbacks and linear readouts can be used to generate different types of limit cycles in a highly robust fashion. Furthermore, we show that such devices are able to produce different gaits with the same morphological structure. Moreover, we demonstrate that such morphological computation devices are capable to implement a smooth input dependent switching between different limit cycles.

4.1 Introduction

Looking at the list of examples of morphological computation in the introduction of chapter 3 one can see that locomotion is 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. So far a standard approach to produce such patterns for locomotion are networks of coupled oscillators. Due to their rather abstract implementation they can be employed for all kinds of locomotion. For example, Righetti and Ijspeert applied them to various quadrupeds [46] or Ijspeert et. al. [21] implemented such a network in order to produce different movements for a lamprey robot. In general these implementations have the advantages that they exhibit robustness and that they are generic, thus can be tailored for a specific task. However, a disadvantage is that the involved parameters are either hand-tuned or found by time intensive nonlinear optimization schemes, e.g., genetic algorithms. Another drawback is that they just produce some abstract trajectories, which then have to be followed by high torque servo motors. Hence, the dynamics of the physical body of the robot are not taken into account and therefore, one still has to deal with previously discussed disadvantages of classical robot design (see Section 3.1). In contrast to that we will propose a morphological computation setup, which employs directly the nonlinear dynamics of the compliant body parts for computation. The setup also exhibits the desired robustness and the generic applicability. Moreover, the adaptable parameters are some linear, static weights, which can be found with simple linear regression. Naturally, this is much faster than any nonlinear optimization scheme and it is guaranteed to get not stuck in local minima.

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 therefore clear insight on how physical bodies can be employed for computational tasks, which need persistent memory. More specific, we will demonstrate that nonlinear mass-spring systems[1], which are typically used to describe the dynamics of compliant biological and robotic body parts, can be enhanced by a nonlinear, but static feedback and a nonlinear readout to emulate in principle any conceivable computation on some analogous input stream. Moreover, we will show, if the dynamics of the compliant body is sufficiently complex, already a linear feedback and a linear readout is sufficient, since the needed nonlinearities are "outsourced" to the physical body. Remarkably, although the presented theory in this chapter is entirely different to the one introduced in Chapter 3, it suggests also, that high dimensionality and nonlinearity, both properties usually undesired due to the difficulties to control them, are desired attributes for computationally powerful 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 computations on input streams. In Section 4.3 we demonstrate, how this theoretical model can be practically implemented with a real physical body. In order to support our theoretical model we provide a number of experiment in Section 4.5. Finally, we close with a conclusion.

4.2 Theoretical Foundations

We present a theoretical model for morphological computation, which is based on a result by Maass et. al. [33]. They proved that certain types 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. Remarkably, the original dynamic system stays unchanged. Only the static feedback drives the system in order to emulate, in conjunction with a static readout, a given nonlinear target system. Maass et. al. applied their theoretical framework to different models of recurrent circuits of neurons and demonstrated that those generic networks gained computational power to carry out computations with persistent memory (even under the influence of noise), when they added a suitable feedback. A remarkable fact, which also provides the basis for our theoretical analysis of morphological computation, is that such a fixed dynamical system, is not required to have a particular form. It is only requested that it belongs to the class S_n of feedback linearizable systems.

[1]Note that they are of the same type as described in Section 3.4

Theorem 4.1. *A system of the form*

$$\mathbf{x}'(t) \quad = \quad f(\mathbf{x}(t)) + g(\mathbf{x}(t)) \cdot v(t) \ , \tag{4.1}$$

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

(LI) *The set of vector fields* $\{\mathbf{g}(\mathbf{x}), ad_{\mathbf{f}}\mathbf{g}(\mathbf{x}), \ldots, ad_{\mathbf{f}}^{n-1}\mathbf{g}(\mathbf{x})\}$ *is linearly independent*[2]

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

in some neighborhood of $\mathbf{x}^0$ *(for proof we refer to [51]).*

Accordingly, a dynamic system of the form of Equation 4.1 is *globally feedback linearizable* if and only if the conditions **(LI)** and **(INV)** hold for the whole state space. We will reserve the letter $\mathcal{C}$ to denote feedback linearizable systems of the form of Equation 4.1, 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 to transform a nonlinear system into a linear system, which is then naturally much easier to control. The corresponding resulting linear system $\mathcal{L}$ to the nonlinear system $\mathcal{C}$ is

$$\mathcal{L}: \quad \mathbf{x}' = \mathbf{A}_n\mathbf{x}(t) + \mathbf{b}_n v(t), \tag{4.2}$$

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 [33, 51]. Such a change of basis can be achieved through an appropriate readout function.

Let us assume that we have a given dynamic system $\mathcal{G}$, which has the order n, is nonlinear, and is of the form

$$\mathcal{G}: \quad z(t)^{(n)} = G(z(t), z(t)', \ldots, z(t)^{(n-1)}) + u(t) \ , \tag{4.3}$$

[2]with $ad_{\mathbf{f}}^{i}\mathbf{g}(\mathbf{x})$ being the i-times recursively applied Lie bracket of f and g (see for example [22]).

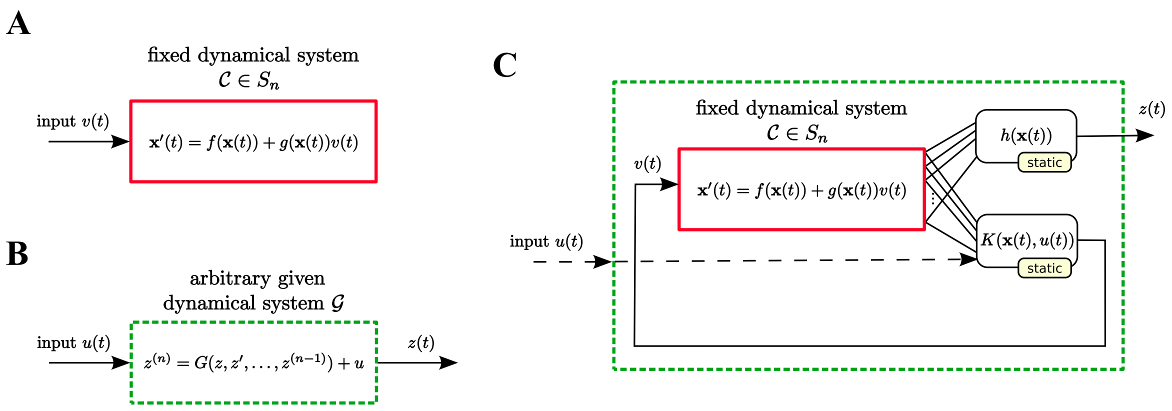


Figure 4.1: Computational architectures considered in Theorem 4.2. **(A)** Fixed dynamical system $\mathcal{C} \in S_n$, which is of the form of Equation 4.1 **(B)** An given arbitrary nth order dynamical system $\mathcal{G}$ (target system) with external input $u(t)$, which should be emulated by system (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 $u(t)$.

where $G : \mathbb{R}^n \rightarrow \mathbb{R}$ is a sufficiently smooth, but otherwise arbitrary, nonlinear function. We can easily demonstrate that the considered system $\mathcal{G}$ is also feedback equivalent to the linear system $\mathcal{L}$ of Equation 4.2. 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. [33]). In other words, for a given nonlinear, dynamic system $\mathcal{G}$ (Equation 4.3) there exist 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 Figure 4.1).

This implies, that one can have a fixed system $\mathcal{C}$, which operates as some universal basic module, and one just has to add a suitable feedback and readout in order to emulate any given nonlinear system $\mathcal{G}$, as long it has the form of Equation 4.3. Note that the description of the system $\mathcal{G}$ in Equation 4.3 is remarkable general. It includes computations like, for example, to describe any time invariant, memory fading operator (i.e., the class of filters, which can be emulated by the morphological computation devices described in Chapter 3), or 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 *contin-*

uous dynamic response to an input stream $u(t)$, hence, one could 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. Remarkably, even under noise the proposed computational setup still has maximal possible computational power within the a priori limitation, i.e., that they can emulate any given finite state machine (FSM) [33]. Note that any digital computer is a FSM.

As already mentioned the system $\mathcal{C}$, the fixed basic module of our computation device, does not have to have a special form, but only has to belong to the class S_n of feedback linearizable systems. This remarkable fact allows us to apply this theory to morphological computation, by employing the type of dynamic systems, which are typically used to described the dynamic behavior of the morphological structure of biological systems (i.e., muscle-skeleton system), as well to describe the dynamic behavior of compliant parts of robots. Consequently, we can employ the dynamic structure of a robot (which is fixed) as such a basic dynamic module $\mathcal{C}$. If we apply appropriate feedbacks and readouts to this physical body, it then gains universal computational power for analog computing. Therefore, one can argue that the dynamic structure can be employed as a part of a morphological computation device. Typically, the considered dynamics are mathematically described by nonlinear mass-spring systems

$$\begin{aligned} x_1' &= x_2 \\ x_2' &= -p(x_1) - q(x_2) + \frac{1}{m}v \ , \end{aligned} \tag{4.4}$$

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[3]. In order to have a physically realistic and therefore stable system these functions have to be monotonically increasing and fulfill $p(0) = 0$ and $q(0) = 0$. In accordance to the previous discussion we can now state following theorem, which will be proved in next section:

Theorem 4.2. *The dynamical system of a nonlinear mass-spring system (Equation 4.4) acquires through feedback universal computation capabilities for analog computing. More precisely, through a proper choice of a (memoryless) feedback function $K(\mathbf{x}(t), u(t))$ and a (memoryless) readout $h(\mathbf{x}(t))$ it can simulate any given dynamical system of the form*

$$z(t)'' = G(z(t), z(t)', z(t)'') + u(t) \ , \tag{4.5}$$

with a sufficiently smooth function G.

The corresponding proofs can be found in Section 4.8. Note that the order of the system

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

4.5 is only two, since the dimension of the original dynamic system (i.e., the mass-spring system of Equation 4.4) is also two. However, as Maass et. al. pointed out even systems consisting of several higher order differential equations of the form Equation 4.3 can be simulated by fixed systems of the form Equation 4.4 with a corresponding number of feedbacks (see Theorem 1 in [33]).
Accordingly, we are able to employ the dynamic properties of compliant body parts, which can be described by Equation 4.4, to emulate any nonlinear, dynamic response to an input stream $u(t)$. This offers us the possibility take advantage of the dynamics of the structure of the robot, just by applying appropriate static feedbacks and static readouts. From this point of view the dynamics of the compliant body parts do not have to be suppressed any longer, but rather can serve as basic dynamic modules (i.e., as a systems $\mathcal{C}$) for a powerful morphological computation device.

.

4.3 Application of the theory to recurrent networks of nonlinear springs and masses

In the previous section (along with the proofs in Section 4.8) has been demonstrated that nonlinear mass-spring systems can gain universal computational power in the previously discussed sense by applying suitable nonlinear feedbacks and nonlinear readouts. A remarkable fact is, that the feedback as well as the readout function, which define what system is emulated, are static. Hence, they provide suitable targets for supervised learning techniques. These static functions could be implemented, for example, by a feedforward neural networks with one hidden layer, since such networks can approximate any continuous function with arbitrary precision [19]. However, this is only possible if the desired feedback and readout functions are known, and this is only possible if the exact properties of the used mass-spring system, i.e., functions $p(x_1)$ and $q(x_2)$, and the mathematical model of the target system are known. In general this is not the case and therefore, no clear target functions are available.

However, 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 of nonlinear combination. For example, some highly complex, nonlinear, dynamic systems could be employed to provide the necessary nonlinearities. For a practical implementation one could consider a recurrent network of nonlinear springs and masses (e.g., of the type of Section 3.4) and employ its inherent complex nonlinear dynamics. Actually, a typical morphological structure, either of a biological organism or a compliant robot, embodies a similar pool of complex dynamics. Therefore, we could move the load of providing nonlinearity directly to the morphological structure. This results in a remarkable consequence. While in most classical approaches in robot control this complexity is unwanted, and is therefore overridden (by the use of rigid body parts and high torque servos), in contrast, our proposed setup requires such high complexity and nonlinearities in the dynamics of the morphological structure. Note that Paul [42] suggested a heuristic that a physical body with a greater amount of "dynamic coupling" (complexity) has a higher possibility of a reduced control requirement. While her findings were based on experimental experience and thought experiments, we provide a rigorous mathematical model to explain the need for complex dynamics of the morphology of robots and biological organisms.

The underlying idea is to see the morphological structure as some fixed, finite and 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. In other words, the morphological structure provides the dynamics and the needed nonlinearities in one physical body. Note that the notion of a kernel, especially the one of a finite kernel, that we use here, is the same as previously discussed in Section 3.1. As a consequence, 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, learning linear readouts is much simpler and faster than to learn any nonlinear function. Moreover, it is guaranteed that learning can not get stuck in local minima of the mean-squared error function.

From this previous line of argumentation we can conclude that the previously introduced random, recurrent networks of nonlinear springs and masses can be used as part of the new morphological computation devices. We simply have to add linear readouts and linear feedbacks in order to emulate nonlinear differential systems of the type of Equation 4.3. Again as before, we will refer to such networks simply as *mass-spring networks.* In the next Section we revise the construction, the simulation and the learning with such mass-spring networks.

4.4 Implementation of mass-spring networks

4.4.1 Constructing mass-spring networks

The construction of the mass-spring networks was very similar the process described in Section 3.4. A fixed number of N nodes (mass points) were randomly positioned (uniformly distributed) within a defined range of a two dimensional plane. Subsequently, we connected these mass points by nonlinear springs. In order to find reasonable, non-crossing spring connections we calculated a Delaunay triangulation on this set of points, resulting in L non-crossing spring connections. A schematic example of such a mass-spring network can be seen in Figure 4.2. Every single nonlinear spring of such a network can be described by Equation 4.4. At the beginning of the simulation we assumed the

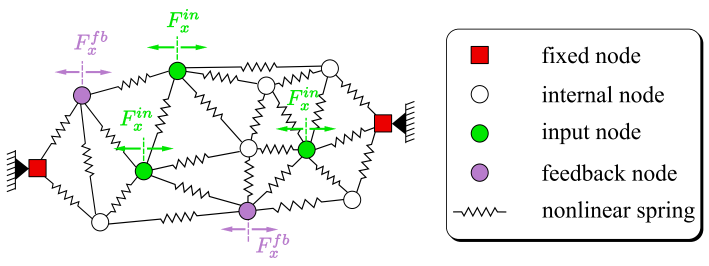


Figure 4.2: Schematic example of a generic mass-spring network. The nodes (masses) are connected by nonlinear springs. The red nodes are fixed in order to hold the network in place. The green nodes are randomly chosen inputs nodes, which receive the input in form of horizontal forces scaled by randomly initiated weights. The purple nodes are feedback nodes. Similarly, they receive the output $y(t)$ in form of a scaled, horizontal force.

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 [4], had to be monotonically increasing and fulfill $p(0) = 0$ and $q(0) = 0$. Typically, they are modeled by 3rd order polynomials [40]. 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 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. The left most and the right most mass nodes were fixed in order to keep the network in place (squared, red nodes in Figure 4.2). A certain percentage of points were randomly chosen to be input nodes (green nodes in Figure 4.2). During simulation they received a linearly scaled version of the current input in form of a horizontal force. Before the simulation started the input scaling factors (weights $\mathbf{w}_{in} = [w_{in,1}, w_{in,2}, \ldots]^T$) had been randomly drawn from a certain range and had been fixed subsequently. 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 $\mathbf{w}_{fb} = [w_{fb,1}, w_{fb,2}, \ldots]^T$. 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

[4]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 [26]).

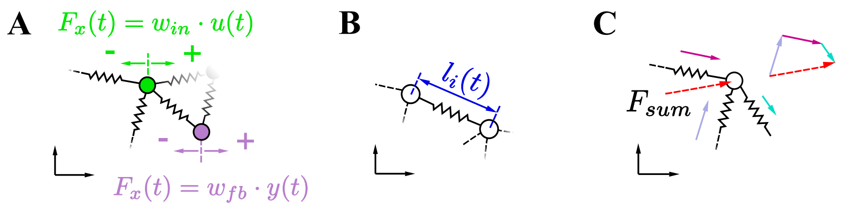


Figure 4.3:

Figure 4.4: Implementation of the input, the feedback, the linear readout and the simulation of the forces at a mass point in a mass-spring networks. **(A)** The input is applied to an input node as a horizontal force F_x proportional to the input signal u (scaled by a randomly initialized weight w_{in} for this input node). The feedback is similarly implemented (with a feedback of an output weighted by w_{fb}). **(B)** The output of the systems is the weighted sum of all L spring lengths $y(t) = \sum_{i=1}^{L} w_{out,i} l_i(t)$. 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.

actual spring lengths $y(t) := \sum_{i=1}^{L} w_{out,i} l_i(t)$. The output weights ($\mathbf{w}_{out} = [w_{out,1}, w_{out,2}, \ldots, w_{out,L}]^T$), 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 $\mathbf{W}_{in}$, $\mathbf{W}_{fb}$ and $\mathbf{W}_{out}$.

4.4.2 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

$$mp_x'' = F_x + w_{in}u + w_{fb}y \tag{4.6}$$

$$mp_y'' = F_y \ , \tag{4.7}$$

where p_x'' and p_y'' were the accelerations of the mass point relative to a global reference frame split up into its two spacial dimensions, F_x and F_y were the forces acting on the mass in the corresponding spacial dimensions, $w_{in}u$ was the weighted input, and $w_{fb}y$ the linear feedback. Note that the input was defined as a horizontal force (see Figure 4.4A) and if the mass point was no input node $w_{in} := 0$, accordingly for the feedback $w_{fb} := 0$. For the sake of simplicity 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 Equation 4.4. The value of x_1 was calculated by the actual length

$l(t)$ (Euclidean distance between the two masses, which the spring connected) and the resting length l_0. The velocity x_2 was approximated by $(x_1(t) - x_1(t - \Delta t))/\Delta t$ with a time step of $\Delta t = 1$ ms. The resulting forces were calculated by the nonlinear functions $p(x_1)$ and $q(x_2)$. This procedure was repeated for all springs connected to the mass. We assumed that these forces acted along their corresponding spring axes. Finally, all spring forces acting on the regarding mass node were summed up (see Figure 4.4C). Subsequently, the resulting force F_{sum} was split up into its two spacial dimensions and added as forces F_x and F_y to Equations 4.6 and 4.7. If the mass point was an input node the current input $u(t)$ was added in form of a scaled horizontal force (see Equation 4.6 and Figure 4.4A). Accordingly, if the mass point received feedback the corresponding force, i.e., $w_{fb}y(t)$, was added too. The new position and velocity of the mass were found by numerically integrating Equations 4.6 and 4.7 (4th 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., $y(t) = \sum_{i=1}^{L} w_{out,i} l_i(t)$ (see Figure 4.4B).

4.4.3 Learning the linear readout of the mass-spring networks

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 $\mathbf{w}_{out} = [w_{out,1}, w_{out,2}, \ldots, w_{out,L}]^T$ were adjusted. The learning process was carried out with open loops. Instead of the real outputs of the network the target signals were fed back. Therefore, the system was forced into the desired operative state by this "teacher" signal. Hence, this setup is referred to as *teacher forcing*. After the learning process the loops were closed, i.e., the actual outputs were fed back, and the system ran freely. 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 either the feedback target signals or the readout with white noise during the teacher forcing process. In the first experiments of Section 4.5.1 we used the first type of noise to explicitly demonstrate the noise to signal ratio and to give the reader an intuition of the amount of noise we used (see Figure 4.5). In all the other experiments we used the second type of noise. However, in general both approaches work for all tasks. As a result of the added noise the found output weights did not simply reproduce the desired target trajectories, but rather were able to do so even under the influence of perturbations. Hence, the learned trajectories were robust. Remarkably, this noise, which, for example, is inherent to any real world learning by demonstration setup, is crucial for the robustness of the learned outputs.

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 $\mathbf{w}_{out}^* = \mathbf{L}^\dagger \mathbf{T}$, 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). Note also that the feedback weights were not changed at all.

4.5 Experiments

The previously presented theory only states that nonlinear mass-spring systems gain computational power, if we add *nonlinear* feedbacks and *nonlinear* readouts. However, as we have pointed out in Section 4.3, if the physical body is complex enough, linear feedbacks and linear readouts might be sufficient. Yet, it is not possible to conclude from our proposed theory if a given dynamic system $\mathcal{G}$ can be emulated sufficiently well enough by a given mass-spring network. In order to demonstrate that our approach is still applicable we present simulations of the mass-spring nets applied to a number of robotic relevant tasks.

All presented simulations were implemented in Matlab and were simulated at a time step of 1 ms.

4.5.1 Generating stable, nonlinear limit cycles with morphological computation

The general description of dynamical systems (Equation 4.3), 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 Section 4.8.1). 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 is to implement such limit cycles by a nonlinear oscillator or a network of such oscillators. They exhibit the property of robustness towards different types of disturbances. However, they just produce some abstract trajectory without taking the dynamics of the robot into account. Moreover, the parameters of such oscillators typically were hand-tuned or were found by computationally intensive nonlinear search algorithms, .e.g., genetic algorithms. On the other 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 dynamic of the morphology, the adaptation problem can be reduced to linear regression.

We chose three different limit cycles as tasks. They are summarized in Table 4.1.

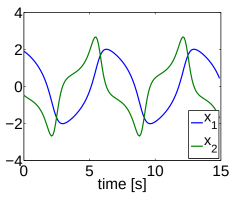
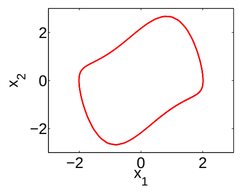
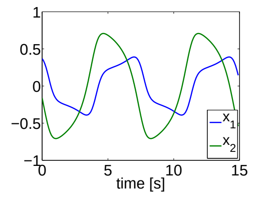
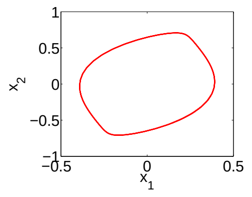
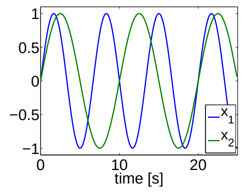
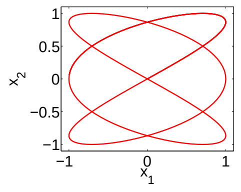
"Van der Pol" $x_1' = x_2$ $x_2' = -x_1 + (1 - x_1^2)x_2$		
"Quadratic limit cycle" $x_1' = x_1 + x_2 - x_1(x_1^2 + x_2^2)$ $x_2' = -2x_1 + x_2 - x_2(x_1^2 + x_2^2)$		
"Lissajous Figure" with $f_1/f_2 = 3/2$ $x_1' = \cos(2\pi 3t)$ $x_2' = \cos(2\pi 2t)$ $t' = 1$		

Table 4.1: Three different limit cycles, which should be generated through morphological computation by generic mass-spring networks 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 x_1 and x_2 (target signals for our tasks). The last column shows the same state variables in the phase plane (i.e., x_1 vs. x_2).

The first limit cycle is the already introduced dynamic system described by the Van der Pol equations. The second one (second row in Table 4.1) is an example taken from [26] (example 2.8). Due to its quadratic terms $(x_1^2 + x_2^2)$ we refer to it as the "quadratic" limit cycle. The third example is an artificial limit cycle 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 Lissajous figure with multiple crossings of the limit cycle trajectory in the state space (see last plot in the third row of Table 4.1). Hence, any dynamical system, which should emulate this system, must have an order higher than two. As already argued in the theory section, although the basic system (the nonlinear mass-spring system) is only two dimensional, due to the right number of feedbacks and/or connections of various basic systems (the latter is the case here), it is possible to emulate systems of higher order too.

We used three different generic mass-spring networks for the three limit cycles. The networks were found by the previously described random process. The number of mass points were $N = 10$ for the Van der Pol and the quadratic limit cycles and $N = 20$ for the Lissajous figure task. The number of the connecting nonlinear springs were 22 (Van

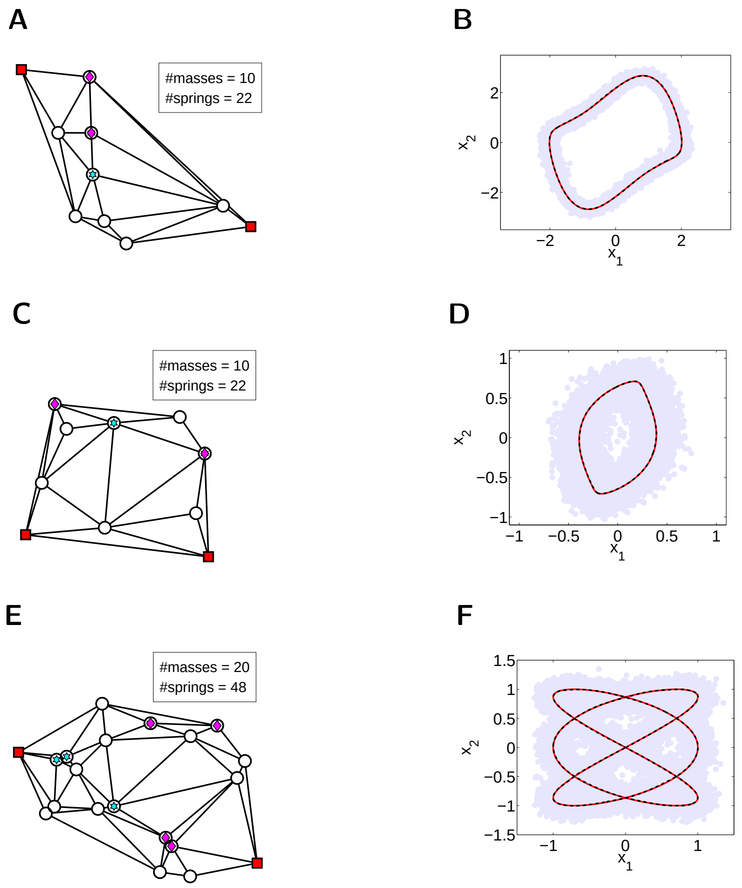


Figure 4.5: Performance of generating different limit cycles of Table 4.1 through morphological computation with linear feedback. The left column shows the used generic mass-spring networks. The red squares depict fixed mass points. The hexagons and the diamonds mark nodes, which received feedback from the first and the second output, respectively. The second column shows the outputs of the systems plotted in phase plane (x_1 vs. x_2), when the systems ran freely , i.e., in closed loop. The red lines are the desired target trajectories, while the black dotted lines are the outputs produced by the morphological computation device. The gray shaded areas depict the regions, which have been covered by the noisy learning data. **(A)** and **(B)** correspond to the "Van der Pol" task, **(C)** and **(D)** to the "quadratic " limit cycle task and **(E)** and **(F)** to the "Lissajous figure" task, as described in Table 4.1.

der Pol), 22 (quadratic limit cycle) and 49 (Lissajous figure). The used mass-spring networks are shown in the first row of Figure 4.5 (A,C and E). The squared, red nodes are fixed mass points. The connections are nonlinear springs. The nodes, which received feedback from the first output (x_1), are marked by purple diamonds. The nodes, which received feedback from the second output (x_2), are marked by aqua marine hexagrams. Note that the feedback nodes were chosen randomly. The corresponding linear feedback weights $\mathbf{W}_{fb}$ were randomly initialized and subsequently fixed. There were no input nodes, since, after learning the morphological computational device should generate autonomously the desired trajectories. The outputs were defined as previously described by a weighted sums of all current spring lengths. The optimal outputs weights were found by the previously described learning process (teacher forcing with additional noise). For more details on the chosen ranges for the various parameter we refer to the supplementary material at our homepage http://www.igi.tugraz.at/helmut/thesis.

After the learning phase the networks were simulated with closed loops, i.e., the outputs were fed back. The right column of Figure 4.5 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. 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 argue that the corresponding real physical bodies can perform such morphological computations too.

As inherent to any simulation there were numerical imprecisions present at any time. Nonetheless, the systems stayed robustly on the trajectories defined by the limit cycles. This was due to the additional noise during the learning process as described in 4.4.3. Therefore, the found output weights did no only represent a "perfect" mapping to the desired outputs, but rather define additionally some region of attraction around those trajectories. We also tested all three networks regarding their stability on the long run. We simulated the networks during 1 million time steps. The networks still stayed robustly on the desired trajectories.

The amplitudes of the noise during learning was quite big compared to the amplitudes of the signals. For example, in the case of the quadratic limit cycle (Figure 4.5D), the noise to signal ratio was more than 0.14 for x_1 and more than 0.25 for x_2. This suggests that the morphological computation device with the learned output weights is not only able to counteract underlying noise with a small amplitude, like of numerical imprecisions, but rather is truly robust towards all kind of disturbances. In order to demonstrate this robustness we conducted various experiments, where we disturbed the system with different types of perturbations. For all robustness experiments we used the network of Figure 4.5A (quadratic limit cycle). However, similar results can be obtained for the other limit cycles too. Figure 4.6 summarizes the results of the conducted robustness experiments.

In a first test (Figures 4.6A and 4.6B) 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 $x_1(t = 10) = 0.09$ was fed back for the next 10 seconds (full blue line). 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 4.6A shows the trajectories of both outputs x_1 and x_2. The red region depicts the time window, when x_1 was locked. Figure 4.6B shows the same trajectories but in a phase plane. Note that the different colors encode the corresponding time windows as labeled in Figure 4.6A. The computational device was able to find back to the desired trajectory after the disturbance had vanished.

In a second robustness test (Figures 4.6C and 4.6D) 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 4.6C and 4.6D show the trajectories of the outputs x_1 and x_2. Again, the used colors in the phase plane correspond to the colors of the labels of the different time windows in Figure 4.6C. 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 "far away" from 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. Therefore, for practical reasons, one could say that the morphological computation device is able to emulate the same limit cycle with a robustness similar to the one in the original system.

A third experiment was conducted in order to show that the setup is also able to recover from stochastic disturbances (Figures 4.6E and 4.6F). 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 its correct trajectories.

In an additional experiment we demonstrate that the system is even able to recover from the state of total rest. Therefore, we started with the network (same network as previously used) 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 additional force would have act on the system (e.g., open loop), the network would have

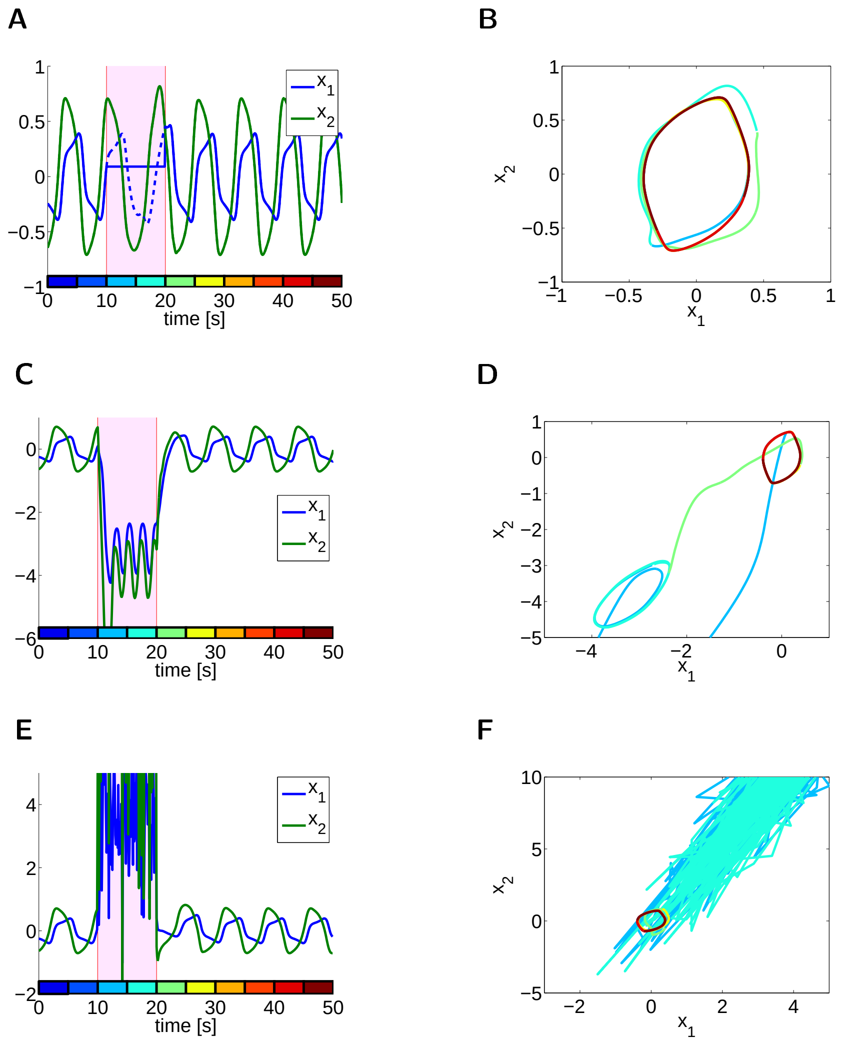


Figure 4.6: Experiments to demonstrate the robustness of the learned limit cycle using morphological computation. The used network was the one depicted in 4.5C ("quadratic" limit cycle). The red regions mark the time windows, when the perturbations appeared. The color coding of the plots of the second 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. 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.

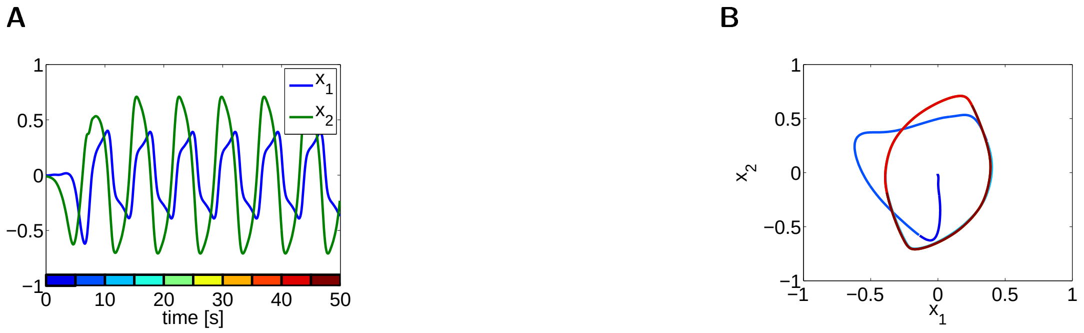


Figure 4.7: The swing up task. The network starts at rest and is able to swing itself up to its nominal limit cycle. Figure **(A)** shows the trajectories over time and **(B)** shows the same trajectories in phase plane. The color coding of (B) corresponds to the color labels of the time windows in (A).

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 Figures 4.7A and 4.7B.

4.5.2 Generation of different walking patterns using the same mass-spring network

So far we have only demonstrated that different, generic mass-spring networks with some linear feedbacks can be used to produce different limit cycles. However, for locomotion, which requires such nonlinear repetitive patterns, it is crucial that the same physical body (i.e., morphological structure) can be employed to produce different patterns. Since the used networks are generic, i.e., they are not constructed for a specific task, they can potentially be used to emulate a number of different nonlinear dynamic systems. Moreover, since the physical body only serves as some basic dynamic module, only the readout and the feedback (both static) define what system is emulated. Hence, different static feedbacks loops can force the same physical body to produce different patterns. Furthermore, since in our setup the feedbacks were randomly initialized and subsequently fixed, already different readout weights $\mathbf{W}_{out}$ are sufficient to produce different dynamic patterns.

The task was to produce four different walking patterns, namely walk, trot, pace and bound, for a generic quadruped with the same morphological structure. Righetti and Ijspeert [46] used a network of four coupled oscillators with four different couplings in order to produce four different gates. We demonstrate that it is possible to produce

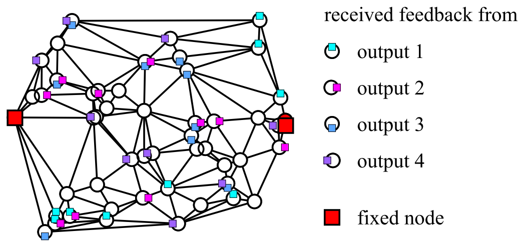


Figure 4.8: Used generic network with four outputs (and feedbacks) for the task to produce different gaits with one physical body. Different colors and relative positions mark different feedbacks.

the same rhythmic patterns with one fixed physical body (i.e., mass-spring network) with some fixed feedback weights, but four different, linear readouts.

The used mass-spring network can be seen in Figure 4.8. It consisted of 50 masses and 137 nonlinear springs. It was constructed as previously described in Section 4.4. For more details on the used values we refer to the supplementary material at our homepage http://www.igi.tugraz.at/helmut/thesis. 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 [46] 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 walking pattern, 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 4.9 summarizes these results. Figure 4.9A shows the four trajectories, which were produced, when the walking gait matrix $\mathbf{W}_{walk}$ 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 (Figure 4.9B) shows the output of the morphological computation device, when the loops at $t = 0$ suddenly were closed with the matrix to produce the trot pattern ($\mathbf{W}_{trot}$) instead of $\mathbf{W}_{walk}$, i.e., the readout switched from walk to trot. After some transition time the system settled to the desired trot pattern. Figure 4.9C and 4.9D show the corresponding responses, when it switched suddenly from walking to pace (C) and walking to bound (D). 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

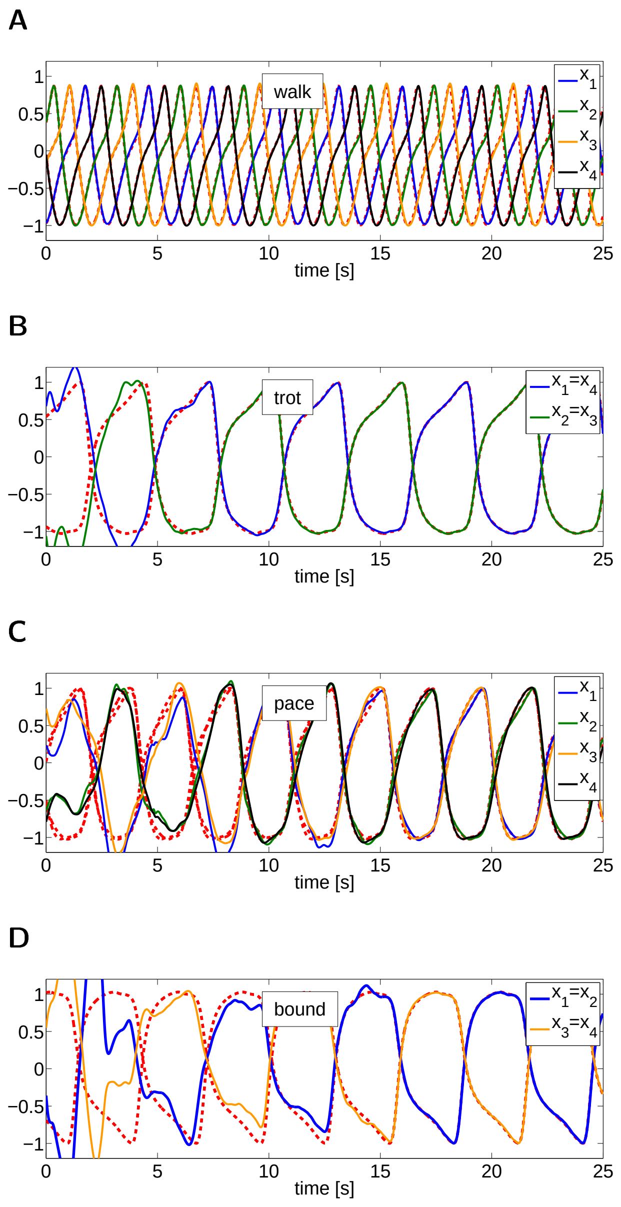


Figure 4.9: Generation of different walking patterns for a generic quadruped robot by the use of one morphological structure. **(A)** walking pattern produce by the morphological computation device.**(B)**, **(C)** and **(D)** responses of the device, when at $t = 0$, the weights of the outputs suddenly changed from $\mathbf{W}_{walk}$ to $\mathbf{W}_{trot}$, $\mathbf{W}_{pace}$ and $\mathbf{W}_{bound}$, respectively. The red dotted lines are the target patterns. In all case, after some transition time, the setup was able to produce the desired patterns.

readouts (or sudden switches between them) were sufficient to force the mass-spring to produce robustly the desired nonlinear patterns.

4.5.3 Generation of different limit cycles depending on an input stream

In the previously presented examples mass-spring networks were employed to generate autonomously different rhythmic output patterns. However, our theory suggests that the proposed setup can emulate even more complex dynamical systems. For example, the proposed morphological computation devices are also able to produce input dependent limit cycles. The input stream could be, for example, some sensory signal, which delivers the information to decide which pattern has to be produced.

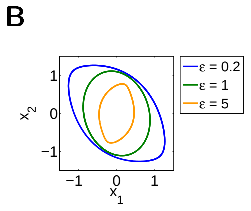
For this task we adapted the previously used equations of the "quadratic" limit cycle (of Table 4.1) by adding a parameter ε (i.e., the input to the system), which gives us the possibility to change smoothly the shape of the limit cycle. Figure 4.10 summarizes the properties of the new target system. Figure 4.10A shows the equations of dynamic 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 Figure 4.10B. 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 Figures 4.10C, 4.10D and 4.10E. Note that the amplitude, the shape as well the frequency change in dependence of the input ε. Figure 4.10F and 4.10G depict this fact by plotting the amplitude and the frequency versus the input ε for the state variable x_1.
Figure 4.11 summarizes the learning data used for the task. Figure 4.11A shows the time-varying input $\varepsilon(t)$ over the whole learning time of 200 s. Figure 4.11B show the resulting limit cycles, i.e., the targets, in phase portrait. The color coding correspond to the colored labels of the time windows in Figure 4.11A.

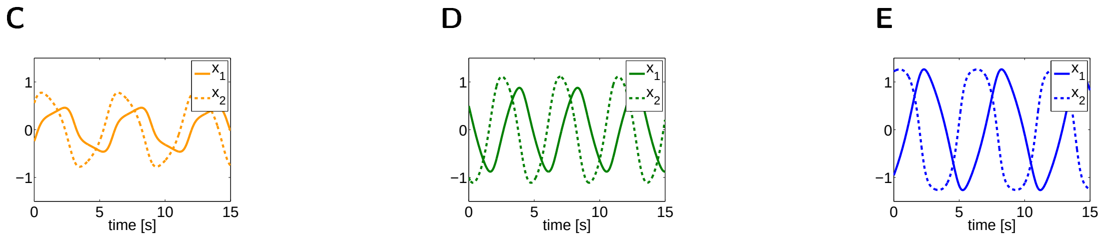
For the simulation we used a mass-spring network with $N = 100$ nodes and $L = 283$ connecting springs. The network was constructed as previously described in Section 4.4. For more details of the used values we refer to the supplementary material at our homepage http://www.igi.tugraz.at/helmut/thesis. For the learning we used the previously described teacher forcing setup with superimposed noise.

Subsequently, the morphological computation device was tested, if it was able reproduce the desired input dependent limit cycles. Figure 4.12 summarizes the results. Note that at the end of the learning process the input was $\varepsilon = 5$ (see Figure 4.11A) and therefore the mass-spring network was in the state to reproduce the corresponding limit cycle. The two plots of Figure 4.12A 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, while the right plot shows the corresponding trajectory in the phase plane. In the second row (Figures 4.12B) the response of the system can be

A

$$\begin{aligned} x_1' &= x_1 + x_2 - \varepsilon \cdot x_1(x_1^2 + x_2^2) \\ x_2' &= -2x_1 + x_2 - x_2(x_1^2 + x_2^2) \end{aligned}$$





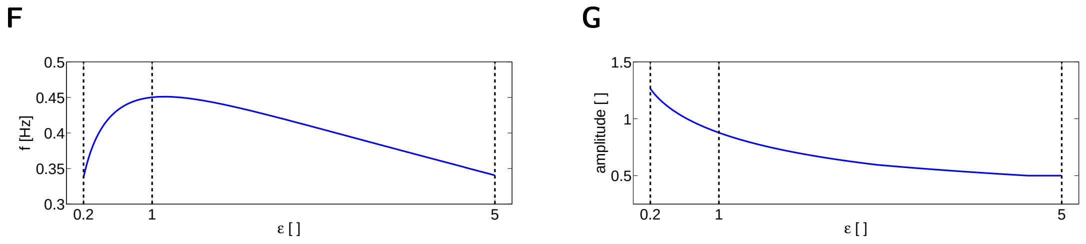


Figure 4.10: Target system for the input depended limit cycle generator task. **(A)** Equation of the dynamical system with the input ε. **(B)** Different limit cycles for the inputs $\varepsilon = 5$, $\varepsilon = 1$ and $\varepsilon = 0.2$ in phase plane. **(C)-(E)** The corresponding trajectories in time. **(F)** and **(G)** change of amplitude and frequency in dependence of the input ε.

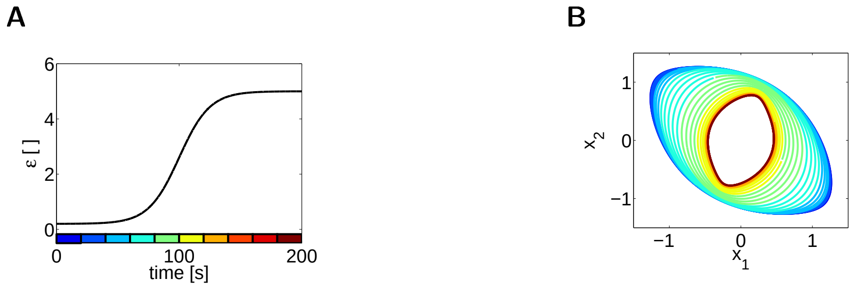


Figure 4.11: Data used to learn to switch smoothly between different limit cycles depending on the input ε. **(A)** Input ε to the system during learning, ranging from 0.2 to 5. The total learning time was 200 s at a time step of 1 ms. The different colored time windows correspond to the color coding in Figure **(B)**, where the resulting changes of the state variables x_1 and x_2 as a result of the changing input ε are plotted.

seen, when the input suddenly changed to a constant value of 1.0. Hence, the system should generate the corresponding limit cycle (compare to Figure 4.10D). After some transition phase, due to the jump in the input from 5 to 1, the system settled down to the desired limit cycle. After that it stayed robustly on the desired trajectories. The last row (Figure 4.12C) shows the response of the network when the input suddenly changes for 5 to 0.2. Again the morphological computation device delivers robustly, after some transition time, the desired limit cycle.

Let us emphasize some points 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, here the same static feedback loop (i.e., readout plus feedback) was used to produce different types of limit cycles. Third, the input was applied as some randomly weighted, but constant forces acting on some randomly chosen input nodes. Such constant input can be seen as *squeezing* the network a certain points (i.e., at the input nodes). Hence, figuratively spoken, the network produced different limit cycles, depending how strong (from 0.1 to 5) it was compressed.

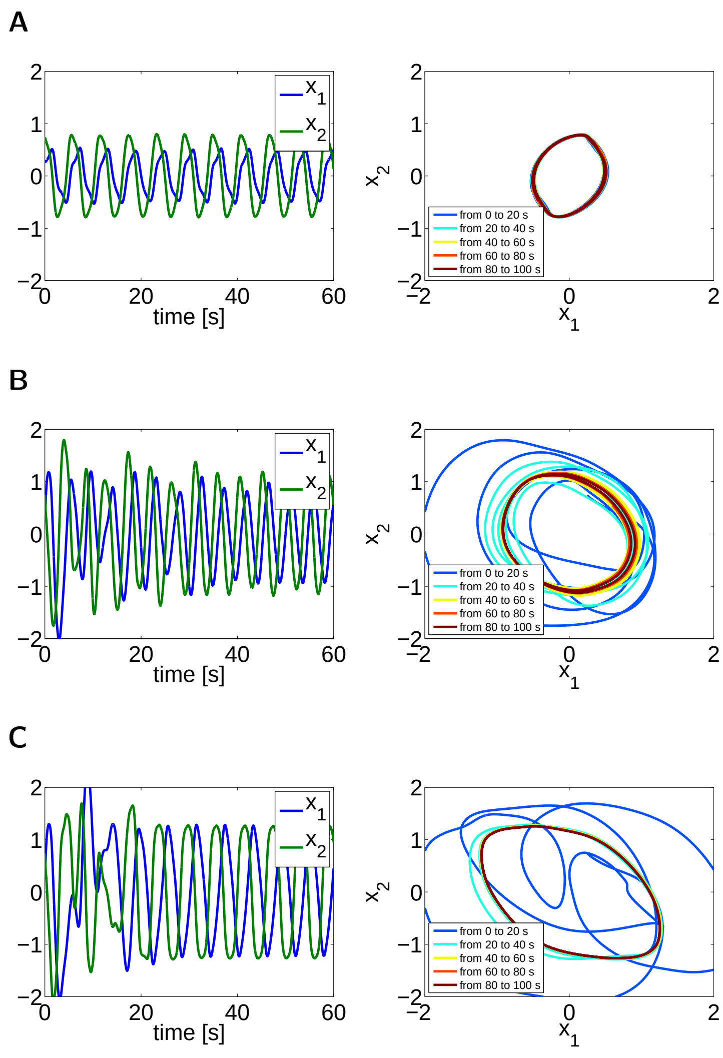


Figure 4.12: Testing the morphological computation device to emulate different limit cycle depending on the input. The left column shows the two output variables variables x_1 and x_2 evolving over time. The right column shows the corresponding trajectories in the phase plane for three different cases. 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 receives instead $\varepsilon = 5$ a constant input of $\varepsilon = 1$. The system changes its output to the corresponding limit cycle. Case (**C**): The network received suddenly an input of 0.2 (instead of 5) and produces, after some transition time, the corresponding limit cycle (compare target limit cycles in 4.11B).

4.6 Discussion

We introduced a new type of theoretical model for morphological computation to overcome the limitations of the previous approach of Chapter 3. We demonstrated that certain types of physical bodies (which have fading memory) can emulate a class of nonlinear systems (which can have even persistent memory) when we add appropriate nonlinear, but static feedbacks and readouts. We argued that, if the physical body is complex enough, it could serve as some finite *kernel*. As a result only linear feedbacks and readouts are sufficient. We demonstrated the validity of this point of view by simulating recurrent networks of nonlinear springs and masses, which can serve as models for complex, compliant body parts of robots and biological systems. Remarkably, already small networks in conjunction with linear feedbacks and readouts were able to represent the dynamic systems of nonlinear limits cycles (which are typically used for locomotion). Furthermore, the learned trajectories were highly robust to all kinds of perturbations. We also demonstrated that the same fixed morphology (i.e., same physical body) can be used to generate, for example, different walking gaits. Finally, we presented results, which showed, that even an input dependent smooth switch between different limits cycles can be emulated by the proposed morphological computation device.

There are a number of remarkable conclusions we can draw from the presented results. First, complexity and compliance of the physical body are crucial in order to be able to outsource parts of the computation to the morphology and therefore be able to use linear readouts and feedbacks. As a consequence the task of learning to emulate complex, nonlinear dynamic systems can then be reduced to finding some static, linear output weights.
Second, the physical body, which is naturally to a high degree fixed, can be employed for different nonlinear computations.
Third, the superimposed noise during the learning process (teacher forcing) is crucial for the robustness of the learned limit cycle. This is remarkable, since noise, inherently present in any real world task, is typically unwanted.
Considering the presented points one can conclude that a transfer of the presented approach from our abstract networks to real physical robots should be possible in a straight forward manner. Additionally, our results could give raise to entirely new types of robots or robot body parts.

4.7 Acknowledgment

This chapter is based on a draft of a paper with the tentative title *"Morphological Computation with Explicit Feedback"*. The authors are myself (HH), Rolf Pfeifer (RP), Rudolf M. Füchslin (RF), Auke J. Ijspeert (AI) and Wolfgang Maass (WM). All of them contributed by giving feedback and by proposing experiments. All of them contributed

by giving feedback and by proposing experiments. The basic impulse came from WM and AI. The writing, the proofs, the implementation of the simulations as well the design of the experiments were done my myself (HH).

4.8 Proofs

In this Section we provide the accompanying proofs.

4.8.1 Computational universality property of a single nonlinear mass-spring system

In this section we demonstrate that a physically realistic, nonlinear mass-spring systems has the previously described computational universality property (i.e., we prove Theorem 4.2). Thus, they can be used as basic systems $\mathcal{C}$ to emulate arbitrary, nonlinear, dynamical systems $\mathcal{G}$ of the form of Equation 4.3. 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 too, i.e., can be used for morphological computation. In order to prove that, we have to demonstrate that the dynamic system of Equation 4.4, which describes such nonlinear mass-spring systems, belongs to the class S_n of feedback linearizable systems. Accordingly to Theorem 4.1 the conditions **LI** and **INV** have to be fulfilled, i.e., the set of vector fields $\{\mathbf{g}(\mathbf{x}), ad_{\mathbf{f}}\mathbf{g}(\mathbf{x}), \ldots, ad_{\mathbf{f}}^{n-1}\mathbf{g}(\mathbf{x})\}$ has to be linearly independent, and the distribution generated by $\{\mathbf{g}(\mathbf{x}), ad_{\mathbf{f}}\mathbf{g}(\mathbf{x}), \ldots, ad_{\mathbf{f}}^{n-2}\mathbf{g}(\mathbf{x})\}$ has to be involutive.

For the case of the nonlinear mass-spring system (Equation 4.4) the order is $n = 2$ and the regarding vector fields are

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

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 $\mathbf{x}$)

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

where $\dot{(.)}$ denotes the first derivative of the regarding state variable. First, we have to show that following set of vector fields

$$[\mathbf{g},\, ad_{\mathbf{f}}\mathbf{g}] = \begin{pmatrix} 0 & \frac{1}{m} \\ \frac{1}{m} & -\frac{1}{m}\dot{q} \end{pmatrix}$$

is linearly independent (condition **LI**), which is true for any value of $\dot{q}$, 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 Equations 4.4 belongs to the class S_n of feedback linearizable systems and therefore has the previously described computational universality property and can be employed as a basic computation module $\mathcal{C}$.

Assuming we have a given target system $\mathcal{G}$, naturally the question arises, what is the corresponding nonlinear feedback $K(\mathbf{x}, u)$ and the nonlinear readout $h(\mathbf{x})$ for the nonlinear mass-spring system to emulate this system? We will answer this question for 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 trajectories as, for example, used for locomotion. The considered differential equations are

$$\begin{aligned} x_1' &= x_2 \\ x_2' &= -x_1 + (1 - x_1^2)x_2. \end{aligned} \tag{4.8}$$

The corresponding function G is found by rewriting the set of two differential equations of order one into one differential 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 the feedback K as well as the readout h are static. Furthermore, note that if only the feedback $K^{\#}(\mathbf{x}, u) = p(x_1) + q(x_2)$ would be applied, the system would be linearized and the resulting linear system would of the form of Equations 4.2 with $n = 2$.

4.8.2 Computational universality property of an array of linear mass-spring systems

Now we consider a set of **linear** mass-spring systems (along with the proof for neural networks equations like (11) in [33], where they used linear systems in parallel too). Assuming our basic module has the following form

$$\begin{aligned} x_1' &= x_2 \\ x_2' &= -kx_1 - dx_2 + v \ , \end{aligned} \tag{4.9}$$

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

$$\begin{aligned}\begin{pmatrix} x_1' \\ x_2' \end{pmatrix} &= \underbrace{\begin{pmatrix} 0 & 1 \\ -k & -d \end{pmatrix}}_{\mathbf{A}_1}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \underbrace{\begin{pmatrix} 0 \\ 1 \end{pmatrix}}_{\mathbf{b}_1} v \\ y &= \underbrace{\begin{pmatrix} 1 & 0 \end{pmatrix}}_{\mathbf{c}_1^T}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}.\end{aligned}$$

We now consider a system, which is made of m different parallel linear mass-spring systems of the form of Equation 4.9, with $x_{i,j}$ being the jth state variable of the ith system ($j = 1,2$ and $i = 1,2,\ldots,m$).

$$\begin{pmatrix} x_{1,1}' \\ x_{1,2}' \\ x_{2,1}' \\ x_{2,2}' \\ \vdots \\ x_{m,1}' \\ x_{m,2}' \end{pmatrix} = \underbrace{\begin{pmatrix} \mathbf{A}_1 & 0 & \cdots & 0 \\ 0 & \mathbf{A}_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{A}_m \end{pmatrix}}_{\mathbf{A}} \begin{pmatrix} x_{1,1} \\ x_{1,2} \\ x_{2,1} \\ x_{2,2} \\ \vdots \\ x_{m,1} \\ x_{m,2} \end{pmatrix} + \underbrace{\begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ \vdots \\ 0 \\ 1 \end{pmatrix}}_{\mathbf{b}} v \tag{4.10}$$

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 4.1). Note that a linear system trivially fulfills the second condition **INV** (see Theorem 6.2 in [50]) 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 $\mathbf{R}$ [50][5]. 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

[5]Actually condition **LI** is the nonlinear counterpart of showing that the nonlinear system is controllable.

a given order n

$$\begin{pmatrix} 0 & 1 & d_i & p(d_{i,}^2) & \cdots & p(d_i^{(n-2)}) \\ 1 & d_i & p(d_{i,}^2) & p(d_{i,}^3) & \cdots & p(d_i^{(n-1)}) \end{pmatrix}, \tag{4.11}$$

where $p(d_i^w)$ denotes a polynomial of d_i of order w.[6] 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 Equation 4.10, 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.

[6]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 is suppressed. The proof holds independently from that for any real positives values of k_i.

Bibliography

[1] A. Alexandrov, A. Frolov, and J. Massion. Axial synergies during human upper trunk bending. *Exp Brain Res*, 118(2):210–220, Jan 1998.

[2] Karl J. Åstrom and Björn Wittenmark. *Adaptive Control*. Addison Wesley Longman, second edition, 1995.

[3] A.F. Atiya and A.G. Parlos. New results on recurrent network training: unifying the algorithms and accelerating convergence. *Neural Networks, IEEE Transactions on*, 11(3):697 –709, May 2000.

[4] P. Baerlocher and R. Boulic. Task-priority formulations for the kinematic control of highly redundant articulated structures. In *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*, volume 1, pages 323–329, 13-17 Oct. 1998.

[5] Paolo Baerlocher and Ronan Boulic. An inverse kinematics architecture enforcing an arbitrary number of strict priority levels. *The Visual Computer*, 20(6):402–417, August 2004.

[6] Peter L. Bartlett and W. Maass. Vapnik-Chervonenkis dimension of neural nets. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 1188–1192. MIT Press (Cambridge), 2nd edition, 2003.

[7] S. Boyd. *Volterra Series: Engineering Fundamentals*. PhD thesis, UC Berkeley, 1985.

[8] S. Boyd and L. Chua. Fading memory and the problem of approximating nonlinear operators with volterra series. *Circuits and Systems, IEEE Transactions on*, 32(11):1150–1161, Nov 1985.

[9] Steve Collins, Andy Ruina, Russ Tedrake, and Martijn Wisse. Efficient Bipedal Robots Based on Passive-Dynamic Walkers. *Science*, 307:1082–1085, February 2005.

[10] A. d'Avella and E. Bizzi. Shared and specific muscle synergies in natural motor behaviors. *Proc Natl Acad Sci*, 102(3):3076–3081, 2005.

[11] Andrea d'Avella, Philippe Saltiel, and Emilio Bizzi. Combinations of muscle synergies in the construction of a natural motor behavior. *Nature*, 6(3):300–308, March 2003.

[12] D. P. Ferris, M. Louie, and C. T. Farley. Running in the real world: adjusting leg stiffness for different surfaces. *Proc Biol Sci*, 265(1400):989–994, Jun 1998.

[13] Sandra M S F Freitas, Marcos Duarte, and Mark L Latash. Two kinematic synergies in voluntary whole-body movements during standing. *J Neurophysiol*, 95(2):636–645, Feb 2006.

[14] Ambarish Goswami. Postural stability of biped robots and the foot rotation indicator (FRI) point. *The International Journal of Robotics Research*, 18(6):523–533, 1999.

[15] Ambarish Goswami and Vinutha Kallem. Rate of change of angular momentum and balance maintenance of biped robot. In *Proceedings of the 2004 IEEEE International Conference on Robotics and Automation ICRA*, volume 4, pages 3785–3790, April 2004.

[16] Helmut Hauser, Gerhard Neumann, Auke J. Ijspeert, and Wolfgang Maass. Biologically inspired kinematic synergies provide a new paradigm for balance control of humanoid robots. In *Proceedings of the 7th IEEE RAS/RSJ Conference on Humanoids Robots (HUMANOIDS07)*, Pittsburgh, PA, December 2007.

[17] Helmut Hauser, Gerhard Neumann, Auke J. Ijspeert, and Wolfgang Maass. Nonlinear kinematic synergies enable linear balance control. *[submitted to] Biological Cybernetics*, 2010.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9:1735–1780, November 1997.

[19] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[20] Fumiya Iida and Rolf Pfeifer. Sensing through body dynamics. *Robotics and Autonomous Systems*, 54(8):631–640, 2006.

[21] A. Ijspeert, A. Crespi, D. Ryczko, and J.-M. Cabelguen. From swimming to walking with a salamander robot driven by a spinal cord model. *Science*, 315(5817):1416–1420, 2007.

[22] Alberto Isidori. *Nonlinear Control Systems*. Springer-Verlag GmbH, third edition, 2001. ISBN: 3-540-19916-0.

[23] S. Kagami, F. Kanehiro, Y. Tamiya, M. Inaba, and H. Inoue. AutoBalancer: An Online Dynamic Balance Compensation Scheme for Humanoid Robots. In B.R. Donald, K. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 329–340. A K Peters Ltd., 2001.

[24] S. Kajita, T. Yamaura, and A. Kobayashi. Dynamic walking control of a biped robot along a potential energy conserving orbit. *Robotics and Automation, IEEE Transactions on*, 8(4):431–438, Aug 1992.

[25] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kazuhito Yokoi, and Hirohisa Hirukawa. The 3D Linear Inverted Pendulum Mode: A simple modeling for a biped walking patttern generation. In *Proceedings of the 2001 IEEEE/RSJ International Conference on Intelligent Robots and Systems, Maui*, pages 239–246, November 2001.

[26] Hassan K. Khalil. *Nonlinear Systems.* Prentice Hall, third edition, 2002.

[27] Jens Kober and Jan Peters. Policy search for motor primitives in robotics. In *Advances in Neural Information Processing Systems 22 (NIPS 2008)*, pages 849–856. MA: MIT Press, 2009.

[28] Benjamin C. Kuo and Farid Golnaraghi. *Automatic Control Systems.* John Wiley & Sons, Inc., 8th edition, 2002.

[29] Sung-Hee Lee and A. Goswami. Reaction mass pendulum (rmp): An explicit model for centroidal angular momentum of humanoid robots. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 4667–4672, 10-14 April 2007.

[30] R. Legenstein, S. A. Chase, A. B. Schwartz, and W. Maass. Functional network reorganization in motor cortex can be explained by reward-modulated Hebbian learning. In *Proc. of NIPS 2009: Advances in Neural Information Processing Systems*, volume 22. MIT Press, 2010. in press.

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

[32] W. Maass and E. D. Sontag. Neural systems as nonlinear filters. *Neural Computation*, 12(8):1743–1772, 2000.

[33] Wolfgang Maass, Prashant Joshi, and Eduardo D. Sontag. Computational aspects of feedback in neural circuits. *PLoS Comput Biol*, 3(1):e165, Jan 2007.

[34] N. Mansard and F. Chaumette. Task sequencing for high-level sensor-based control. *Robotics, IEEE Transactions on*, 23(1):60–72, Feb. 2007.

[35] C. R. Mason, J. E. Gomez, and T. J. Ebner. Hand synergies during reach-to-grasp. *J Neurophysiol*, 86(6):2896–2910, Dec 2001.

[36] Tad McGeer. Passive dynamic walking. *Int. J. Rob. Res.*, 9(2):62–82, 1990.

[37] O. Michel. Webots: Professional mobile robot simulation. *Journal of Advanced Robotics Systems*, 1:39–42, 2004.

[38] Ferdinando A. Mussa-Ivaldi. Modular features of motor control and learning. *Current Opinion in Neurobiology*, 9:713–717, 1999.

[39] Alan V. Oppenheim and Alan S. Willsky. *Signal and Systems.* Prentice-Hall Inc., Englewood Cliffs, 1992.

[40] William J. III Palm. *Modeling, Analysis, and Control of Dynamic Systems.* John Wiley & Sons, Inc, 2nd edition, July 1999. ISBN 0-471-07370-9.

[41] C. Paul, F.J. Valero-Cuevas, and H. Lipson. Design and control of tensegrity robots for locomotion. *IEEE Transactions on Robotics*, 22(5):944–957, Oct. 2006.

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

[43] Rolf Pfeifer and Josh C. Bongard. *How the body shapes the way we think.* The MIT Press, 2006.

[44] Rolf Pfeifer, Max Lungarella, and Fumiya Iida. Self-organization, embodiment, and biologically inspired robotics. *Science*, 318:1088–1093, November 2007.

[45] Marko B. Popović, Ambarish Goswami, and Hugh Herr. Ground Reference Points in Legged Locomotion: Definitions, Biological Trajectories and Control Implications. *International Journal of Robotics Research*, 24(12):1013–1032, December 2005.

[46] L. Righetti and A.J. Ijspeert. Pattern generators with sensory feedback for the control of quadruped locomotion. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 819–824, 19-23 May 2008.

[47] Angelo M Sabatini. Identification of neuromuscular synergies in natural upper-arm movements. *Biol Cybern*, 86(4):253–262, Apr 2002.

[48] L. Sciavicco and B. Siciliano. *Modelling and Control of Robot Manipulators.* Springer, 2nd edition, 1999.

[49] YoonSik Shim and Phil Husbands. Feathered flyer: Integrating morphological computation and sensory reflexes into a physically simulated flapping-wing robot for robust flight manoeuvre. In F. Almeida e Costa et al, editor, *ECAL*, pages 756–765. Springer Berlin / Heidelberg, 2007.

[50] Jean-Jacques E. Slotine and Weiping Li. *Applied Nonlinear Control.* Prentice Hall, first edition, 1991.

[51] Eduardo D. Sontag. *Mathematical Control Theory: Deterministic Finite Dimensional Systems.* Springer-Verlag, second edition, 1998. ISBN: 0-387-98489-5.

[52] Russ Tedrake, Teresa Weirui Zhang, and H. Sebastian Seung. Learning to Walk in 20 Minutes. In *Proceedings of the Fourteenth Yale Workshop on Adaptive and Learning Systems*, Yale University, New Haven, CT, 2005., 2005.

[53] Gelsy Torres-Oviedo and Lena H Ting. Muscle synergies characterizing human postural responses. *J Neurophysiol*, 98(4):2144–2156, Oct 2007.

[54] Veronique Tricon, Armande Le Pellec-Muller, Nicolas Martin, Serge Mesure, Jean-Phillipe Azulay, and Sylvie Vernazza-Martin. Balance control and adaptation of kinematic synergy in aging adults during forward trunk bending. *Neuroscience Letter*, 415(1):81–86, Mar 2007.

[55] V. N. Vapnik. *Statistical Learning Theory.* John Wiley (New York), 1998.

[56] Miomir Vukobratović and Branislav Borovac. Zero-Moment Point - Thirty Five Years of its Life. *International Journal of Humanoid Robotics*, 1:157–173, 2004.

[57] Yun Wang, Vladimir M Zatsiorsky, and Mark L Latash. Muscle synergies involved in shifting the center of pressure while making a first step. *Exp Brain Res*, 167(2):196–210, Nov 2005.

[58] D.E. Whitney. Resolved motion rate control of manipulators and human prostheses. *Man-Machine Systems, IEEE Transactions on*, 10(2):47 –53, june 1969.

[59] D. A. Winter. Human balance and posture control during standing and walking. *Gait and Posture*, 3(4):193–214, December 1995. Review article.

[60] L. Wiskott and T. J. Sejnowski. Slow feature analysis: unsupervised learning of invariances. *Neural Computation*, 14(4):715–770, 2002.

[61] M. Wisse and J. Van Frankenhuyzen. 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)*, 2003.

[62] R.J. Wood. Design, fabrication, and analysis of a 3dof, 3cm flapping-wing mav. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 1576–1581, Oct. 29 2007-Nov. 2 2007.

[63] Marc Ziegler, Fumiya Iida, and Rolf Pfeifer. "cheap" underwater locomotion: Roles of morphological properties and behavioural diversity. In *CLAWAR*, 2006.